Jürgen Kehrel

Apple-Assembler lernen

Band 2: Nutzung besonderer Apple-Eigenschaften

Hüthig

6502/65C02

Jürgen Kehrel · Apple-Assembler lernen

Jürgen Kehrel

Apple-Assembler lernen

Band 2: Nutzung besonderer Apple-Eigenschaften

Dr. Alfred Hüthig Verlag Heidelberg

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
»Begleitdiskette zu Apple-Assembler lernen, Bd. 2«
ISBN 3-7785-1244-7

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Kehrel, Jürgen:
Apple-Assembler lernen / Jürgen Kehrel. – Heidelberg: Hüthig

Bd. 2. Nutzung besonderer Apple-Eigenschaften.
[Hauptbd.]. – 1986.
ISBN 3-7785-1170-X

Printed in Germany
Satz, Druck und Bindung: Laub GmbH, Elztal-Dallau

0. Vorwort

„Apple-Assembler lernen" ist ein kompletter Kursus über die Assembler-Programmierung des 6502 und 65C02 auf dem Apple II. Im ersten Band lernten Sie sämtliche Maschinenbefehle und wichtige Grundalgorithmen. Der Umgang mit dem Assembler ASSESSOR wurde geübt. Durch den Simulator IDUS, der sich zusammen mit ASSESSOR auf der Begleitdiskette zum ersten Band befindet, konnten Sie sich Ihre Programme in Zeitlupe ansehen und dabei viel durch Anschaulichkeit lernen.

Mit diesen Grundkenntnissen sind Sie jetzt in der Lage, komplexere Aufgaben zu lösen. Der zweite Band stellt Programme und Unterroutinen vor, um fast alle fundamentalen Probleme auf dem Apple in Maschinensprache zu lösen. Im Gegensatz zu den vielen kurzen Lektionen des ersten Bandes werden wir diesmal weit weniger Schritte ausführen. Die einzelnen Beispiele sind dafür aber auch bedeutend komplizierter, denn fast alle Programme dieses Bandes haben für sich eine sinnvolle Nutzanwendung. Wir werden Musik machen, Fenster und Schrift in hochauflösender Grafik bearbeiten, Textfiles rasend schnell einlesen, die Möglichkeiten des Applesoft durch einen echten Overlay-Manager erweitern und als Krönung der Bemühungen ein Kopierprogramm schreiben, das auf einem 64K-Apple in 32 Sekunden eine Diskette inclusive Formatierung und Verify kopiert.

Wie Sie schon gemerkt haben, ist die Assembler-Programmierung keine Beschäftigung, die man durch reines Zusehen lernt. Sie müssen selbst Programme schreiben, wenn Sie diese Sprache beherrschen wollen. Probieren Sie die Beispiele dieses Buches aus. Verändern Sie sie, experimentieren Sie damit, schreiben Sie ähnliche Programme. Wenn Ihre Fertigkeiten steigen, werden Sie Möglichkeiten finden, das eine oder andere Programm noch zu verbessern. Ideen sind nicht nur Eingabe, sie beruhen auch auf Arbeit und Erfahrung. Mit dem Durcharbeiten des ersten Bandes haben Sie Ihr „Freischwimmer-Zeugnis"

in Assembler erworben. Nach dem zweiten Band haben Sie den „Fahrtenschwimmer" errungen. Sie sind jetzt in der Lage, sich allein auf den vielen Pfaden der Apple-Assembler-Programmierung zurechtzufinden und vielleicht noch höhere Auszeichnungen zu ernten.

Diese zwei Bände sind mit der tatkräftigen Unterstützung meiner Frau Christiane zustande gekommen, die nicht nur viele Fehler aufgespürt und mich darüberhinaus aufgerichtet hat, sondern die über anderthalb Jahre das meist nachmitternächtliche Arbeitsende mit erleiden mußte. Ihr gilt mein besonderer Dank.

Heidelberg, April 1986 Dr. Jürgen B. Kehrel

Inhalt

1. Eine Scheibe bleibt nicht matt

Es gibt nur ganz wenige nützliche Programme, die ohne einen Dialog mit dem Benutzer auskommen. Dabei steht der Bildschirm als Ausgabemedium an erster Stelle. Ein Programm begrüßt Sie, stellt Ihnen Fragen, macht Ihnen Bedienungsvorschläge oder teilt Ihnen Fehlermeldungen mit. Ihre Tastatureingaben werden normalerweise geechot, damit Sie Ihre Antworten verfolgen und korrigieren können. Die Qualität eines Programmes hängt – abgesehen von seinen „inneren" Werten – ganz wesentlich davon ab, wie erfolgreich es seine Bildschirmausgaben bewältigt. Wir werden uns deshalb zunächst damit beschäftigen, wie wir gezielt Texte an jede beliebige Stelle des Bildschirms bringen können.

1.1 Ordnung im Zeilenchaos

Der Apple kann auf seinem Textbildschirm 24 Zeilen (0 - 23) mit je 40 Zeichen (0 - 39) darstellen, also insgesamt 960 Zeichen. Es existieren zwei Schirmseiten, von denen normalerweise nur die Seite 1 angezeigt und benutzt wird. (Das Wort „Seite" wird hier in einer anderen Bedeutung benutzt als bei dem Wort „Speicherseite", die 256 Bytes umfaßt.) Im ROM des Apple sind keine Routinen für die Seite 2 vorhanden, so daß ihre Nutzung nur durch selbstgeschriebene Assembler-Routinen möglich ist. Da dazu der Aufwand meist größer als der Erfolg ist, führt die Textseite 2 ein Mauerblümchen-Dasein.

Jedem Zeichen auf dem Bildschirm entspricht ein Byte im RAM des Apple. Die Seite 1 benutzt Speicherstellen von $0400 bis $07FF, die Seite 2 von $0800 bis $0BFF. Jedes Zeichen wird durch seinen ASCII (American Standard Code for Information Interchange)-Wert repräsentiert. Im Anhang finden Sie eine Tabelle, die Ihnen den Zusammenhang zwischen ASCII-Code und dargestelltem

Zeichen zeigt. Die Attribute NORMAL, INVERSE und FLASH sind beim Apple abweichend von der ASCII-Norm mit in jedes Byte kodiert worden. Beim Apple IIe und IIc sind ferner noch zwei umschaltbare Zeichensätze vorhanden, die bei dem selben Code teilweise andere Bildschirmdarstellungen zur Folge haben. Auch hier zeigt Ihnen eine Tabelle im Anhang die Zusammenhänge.

Die Organisation des Bildschirmspeichers ist beim Apple nicht ganz einfach, auf den ersten Blick sogar chaotisch. Aufeinanderfolgende Zeichen innerhalb einer Zeile stehen auch hintereinander im RAM. Aufeinanderfolgende Zeilen stehen aber leider nicht auch aufeinanderfolgend im Speicher. Vielmehr wird folgendes Muster verwendet:

Zeile	Adresse	Zeile	Adresse	Zeile	Adresse	Scratch
0	$0400-427	8	$0428-44F	16	$0450-477	$0478-47F
1	$0480-4A7	9	$04A8-4CF	17	$04D0-4F7	$04F8-4FF
2	$0500-527	10	$0528-54F	18	$0550-577	$0578-57F
3	$0580-5A7	11	$05A8-5CF	19	$05D0-5F7	$05F8-5FF
4	$0600-627	12	$0628-64F	20	$0650-677	$0678-67F
5	$0680-6A7	13	$06A8-6CF	21	$06D0-6F7	$06F8-6FF
6	$0700-727	14	$0728-74F	22	$0750-777	$0778-77F
7	$0780-7A7	15	$07A8-7CF	23	$07D0-7F7	$07F8-7FF

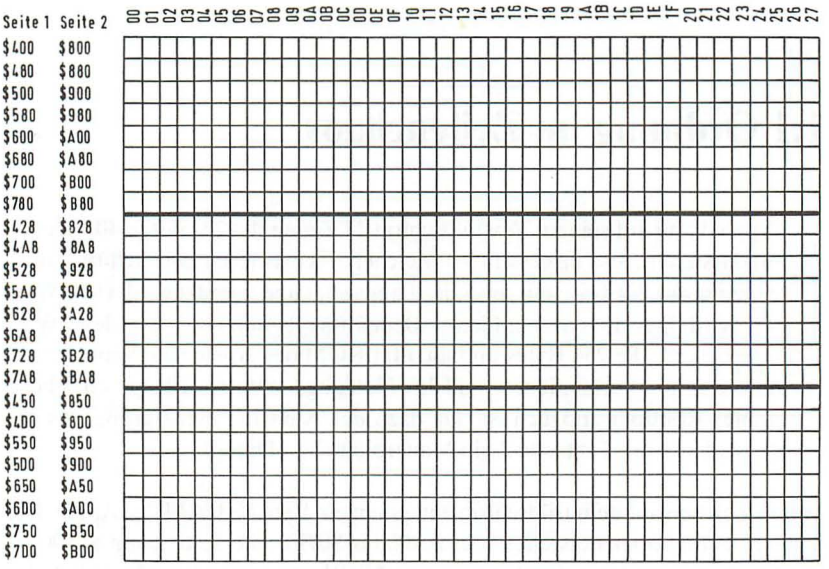


Abb. 1: Adressierung der Textseiten 40Z/Z

Wenn Sie horizontal durch diese Tabelle gehen, so sehen Sie, daß immer 7 Zeilen übersprungen werden. Zu $0400 bis $047F gehören also die Zeilen 0, 8 und 16. Es bleiben 8 Bytes übrig, die nicht gesehen werden können. Sie sind als sogenannte „Scratchpad"-Stellen (kurzzeitige Zwischenspeicher) den Erweiterungskarten in den Steckleisten (Slots) 1 bis 7 und dem DOS zugeordnet. In einigen Assembler-Büchern werden Programme vorgestellt, die blitzschnell den Bildschirm löschen, indem der ganze Speicherbereich von $0400 bis $07FF mit Leerzeichen vollgeschrieben wird. Wundern Sie sich bei solchen „Brutalprogrammen" nicht, wenn hinterher Ihr DOS nicht mehr richtig läuft.

Um Text auf den Bildschirm zu schreiben, müssen Sie die ASCII-Werte der Zeichen in die passenden Speicherstellen schreiben. Im Monitor des Apple existiert eine Routine BASCALC ($FBC1), die die Anfangsadresse jeder Zeile berechnet, wenn ihr beim Aufruf im Akkumulator die Zeilennummer übergeben wird. Das Ergebnis wird in den Zero-Page-Speicherstellen BASL und BASH ($0028 und $0029) abgelegt. Durch die indirekte indizierte Adressierung mit dem Y-Register können Sie dann auf jedes Byte der Zeile zugreifen:
STA (BASL),Y mit Y = 0 ... $27.

BASCALC $FBC1
Berechnet Basisadresse einer Textzeile
(Fenster werden nicht berücksichtigt)

Eingabe:
Akku = Bildschirmzeile

Ausgabe:
BASL ($0028) = Basisadresse Lo
BASH ($0029) = Basisadresse Hi

Die Benutzung von BASCALC ist recht bequem, für manche Fälle aber zu langsam. Dann müssen Sie mit Tabellen arbeiten. Bei unseren HIRES-Programmen werden Sie ein Beispiel für diese Technik finden.

1.2 Der Niedergang der Zeilen

Schon von BASIC aus werden Sie das Textfenster kennengelernt haben. Die Größe des Fensters wird durch die Werte in den Speicherstellen $0020 bis $0023 bestimmt.

```
$0020 = WNDLFT = linker Rand ($00-$27)
$0021 = WNDWDTH = Fensterbreite ($00-$28)
WNDLFT + WNDWDTH <= $28 (IMMER !!)
$0022 = WNDTOP = oberer Rand ($00-$18)
$0023 = WNDBTM = unterer Rand ($00-$18)
```

Es gibt keine Routinen, um diese Parameter auf bestimmte Werte zu setzen. Sie müssen die Speicherstellen selber setzen und auf die Einhaltung der zulässigen Wertebereiche achten. Lediglich zum Setzen des vollen Textfensters gibt es die Monitor-Routine SETTXT ($FB39), die zusätzlich von Grafik auf Text umschaltet. Die vorher aktive Seite bleibt erhalten. Aus HGR2 wird so TEXT2!

SETTXT $FB39
Schaltet auf TEXT und voll geöffnetes Fenster
Setzt Basisadresse BASL/H auf Zeile $17 (23)

Eingabe: –

Ausgabe:
WNDLFT = $00, WNDWDTH = $28, WNDTOP = $00
WNDBTM = $18, CV ($0025) = $17, BASL/H = $07D0

Dem Applesoft-Befehl „TEXT“ entspricht die Routine INIT ($FB2F), die immer auf die Textseite 1 und auf das volle Fenster stellt.

INIT $FB2F
Schaltet auf TEXT1 und voll geöffnetes Fenster
Setzt Basisadresse BASL/H auf Zeile $17 (23)

Eingabe: –

Ausgabe:
WNDLFT = $00, WNDWDTH = $28, WNDTOP = $00
WNDBTM = $18, CV = $17, BASL/H = $07D0

Um das aktive Textfenster eine Zeile nach oben zu rollen (scroll), benutzen wir SCROLL ($FC70).

SCROLL $FC70
Rollt Textfenster eine Zeile nach oben

Eingabe: –

Ausgabe:
BASL/H = unterste Zeile, Y = WNDWDTH

Um das Textfenster eine Zeile nach unten zu rollen, müssen wir eine eigene Routine schreiben.

DOWN-SCROLL

```
1    ;************************
2    ; Scroll-down für 40Z/Z *
3    ;************************
4    ;
5    WNDWDTH   EQU $21
6    WNDTOP    EQU $22
7    WNDBTM    EQU $23
8    BASL      EQU $28
9    BASH      EQU $29
10   BAS2L     EQU $2A
11   BAS2H     EQU $2B
12   VTAB      EQU $FC22
13   VTABZ     EQU $FC24
```

```
                        14  CLEOLZ   EQU $FC9E
                        15  ;
                        16           ORG $300
                        17  ;
0300: A5 23             18           LDA WNDBTM     ;Untergrenze
0302: 38                19           SEC
0303: E9 01             20           SBC #$01       ;Zielzeile
0305: 48                21           PHA            ;merken
0306: 20 24 FC          22           JSR VTABZ      ;Adresse berechnen
0309: A5 28             23  L0       LDA BASL       ;Adresse Lo
030B: 85 2A             24           STA BAS2L      ;umkopieren
030D: A5 29             25           LDA BASH       ;Adresse Hi
030F: 85 2B             26           STA BAS2H      ;s.o. als Ziel
0311: A4 21             27           LDY WNDWDTH    ;Fensterbreite
0313: 88                28           DEY            ;-1
0314: 68                29           PLA            ;Zeilennummer
0315: 38                30           SEC
0316: E9 01             31           SBC #$01       ;Quellzeile
0318: 90 11             32           BCC CLEAR      ;fertig
031A: C5 22             33           CMP WNDTOP     ;Obergrenze Fenster
031C: 90 0D             34           BLT CLEAR      ;fertig
031E: 48                35           PHA            ;Z.nummer merken
031F: 20 24 FC          36           JSR VTABZ      ;Adresse berechnen
0322: B1 28             37  L1       LDA (BASL),Y   ;Quellbyte
0324: 91 2A             38           STA (BAS2L),Y  ;Zielbyte
0326: 88                39           DEY            ;nächstes Byte
0327: 10 F9             40           BPL L1         ;alle Bytes fertig?
0329: 30 DE             41           BMI L0         ;Ja, nächste Zeile
                        42  ;
                        43  ; oberste Zeile löschen
                        44  ;
032B: A0 00             45  CLEAR    LDY #$00       ;ganze Zeile
032D: 20 9E FC          46           JSR CLEOLZ     ;löschen
0330: 4C 22 FC          47           JMP VTAB       ;Zeiger-Update
```

Von unten beginnend wird jeweils die Adresse der letzten (BASL2) und vorletzten Zeile (BASL) berechnet. Die vorletzte Zeile wird in die letze kopiert. Sind wir am oberen Fensterrand angekommen, wird die oberste Zeile ganz (Y = 0) gelöscht.

Unser Programm benutzt noch drei Monitor-Routinen, die wir bisher nicht besprochen haben. VTABZ ($FC24) berechnet die Basisadresse der Zeile im Akkumulator nach BASL/H, wobei das linke Textfenster (WNDLFT) mit eingerechnet wird. VTAB ($FC22) macht dasselbe, nur wird die Zeilennummer aus CV ($0025, vertikaler Cursor) genommen. CLEOLZ ($FC9E) löscht die aktuelle Bildschirmzeile (in BASL/H) ab der Position Y nach rechts.

VTAB $FC22
Vertikalen Tabulator setzen unter Berücksichtigung des linken Fensterrandes

Eingabe:
CV = gewünschter VTAB-Wert ($00-$17)

Ausgabe:
BASL/H = Basisadresse der Zeile

VTABZ $FC24
Vertikalen Tabulator setzen unter Berücksichtigung des linken Fensterrandes

Eingabe:
Akku = gewünschter VTAB-Wert ($00-$17)

Ausgabe:
BASL/H = Basisadresse der Zeile

CLEOLZ $FC9E
Löscht bis zum Ende der aktuellen Bildschirmzeile.

Eingabe:
Y-Reg = horiz. Startwert

Ausgabe:
Y-Reg = WNDWDTH, Akku = $A0

Der Vollständigkeit halber seien an dieser Stelle bereits sehr wichtige Monitor-Routinen genannt:

HOME $FC58
Löscht das aktuelle Textfenster und setzt den Cursor in die Ecke oben links.

Eingabe: –

Ausgabe:
CH = $00, CV = WNDTOP, Akku = BASL, Y-Reg = $00

CLREOP $FC42
Löscht das aktuelle Textfenster von der Cursorposition bis zum unteren Ende. Die Cursorposition bleibt erhalten.

Eingabe: –

Ausgabe: Akku = BASL

1.3 Der Vorhang fällt

Da der HOME-Befehl das Textfenster beachtet, lassen sich mit seiner Hilfe einige effektvolle Löschroutinen schreiben. Damit der Vorhang langsam fällt, müssen wir noch eine Verzögerungsschleife einbauen. Der Monitor besitzt dazu die WAIT-Routine ($FCA8), die über den Akkumulatorinhalt gesteuert wird.

WAIT $FCA8
Zeitverzögerung $13 + {}^{27}/_{2} * A + {}^{5}/_{2} * A^2$ usec

Eingabe: Akku = Verzögerungswert

Ausgabe: Akku = $00

TEXTLOESCH1

```
                    1   ;*******************************
                    2   ;   Textbildschirm   löschen   *
                    3   ;*******************************
                    4   ;
                    5   ; langsam von links nach rechts
                    6   ;
                    7   ;
                    8             ORG $300
                    9   ;
                   10   WAIT      EQU $FCA8
                   11   HOME      EQU $FC58
                   12   WNDWDTH   EQU $21
                   13   ;
0300: A2 01        14             LDX #$01       ;links beginnen
0302: 86 21        15   LOOP      STX WNDWDTH    ;Textfenster
0304: A9 C0        16             LDA #$C0       ;Pause
0306: 20 A8 FC     17             JSR WAIT
0309: 20 58 FC     18             JSR HOME       ;löschen
030C: E8           19             INX            ;nächste Spalte
030D: E0 29        20             CPX #$29       ;zu Ende?
030F: 90 F1        21             BLT LOOP       ;Nein, weiter
0311: 60           22             RTS            ;das war's
```

TEXTLOESCH2

```
                    1   ;*******************************
                    2   ;   Textbildschirm   löschen   *
                    3   ;*******************************
                    4   ;
                    5   ; langsam von unten nach oben
                    6   ;
                    7   ;
                    8             ORG $300
                    9   ;
                   10   WAIT      EQU $FCA8
                   11   HOME      EQU $FC58
                   12   WNDTOP    EQU $22
                   13   ;
0300: A2 17        14             LDX #$17       ;unten beginnen
0302: 86 22        15   LOOP      STX WNDTOP
0304: A9 C0        16             LDA #$C0       ;Pause
0306: 20 A8 FC     17             JSR WAIT
0309: 20 58 FC     18             JSR HOME       ;löschen
030C: CA           19             DEX            ;nächste Zeile
030D: 10 F3        20             BPL LOOP       ;oben angekommen?
030F: 60           21             RTS            ; Ja, fertig
```

TEXTLOESCH3

```
                        1   ;*******************************
                        2   ;   Textbildschirm   löschen   *
                        3   ;*******************************
                        4   ;
                        5   ; langsam von oben nach unten
                        6   ;
                        7   ;
                        8           ORG $300
                        9   ;
                        10  WAIT    EQU $FCA8
                        11  HOME    EQU $FC58
                        12  WNDBTM  EQU $23
                        13  ;
0300: A2 00             14          LDX #$00        ;oben beginnen
0302: 86 23             15  LOOP    STX WNDBTM
0304: A9 C0             16          LDA #$C0        ;Pause
0306: 20 A8 FC          17          JSR WAIT
0309: 20 58 FC          18          JSR HOME        ;löschen
030C: E8                19          INX             ;nächste Zeile
030D: E0 19             20          CPX #$19        ;fertig?
030F: 90 F1             21          BLT LOOP        ;Nein, weiter
0311: 60                22          RTS             ;zurück
```

TEXTLOESCH4

```
                        1   ;*******************************
                        2   ;   Textbildschirm   löschen   *
                        3   ;*******************************
                        4   ;
                        5   ; langsam von rechts nach links
                        6   ;
                        7   ;
                        8           ORG $300
                        9   ;
                        10  WAIT    EQU $FCA8
                        11  HOME    EQU $FC58
                        12  WNDWDTH EQU $21
                        13  WNDLFT  EQU $20
                        14  ;
0300: A2 27             15          LDX #$27        ;rechts beginnen
0302: 86 20             16  LOOP    STX WNDLFT
0304: 38                17          SEC
0305: A9 28             18          LDA #$28        ;Fensterbreite
0307: E5 20             19          SBC WNDLFT      ;anpassen
0309: 85 21             20          STA WNDWDTH
030B: A9 C0             21          LDA #$C0        ;Pause
030D: 20 A8 FC          22          JSR WAIT
0310: 20 58 FC          23          JSR HOME        ;löschen
0313: CA                24          DEX             ;nächste Spalte
0314: 10 EC             25          BPL LOOP        ;fertig?
0316: 60                26          RTS             ;Ja zurück
```

Das kleine BASIC-Programm TL.DEMO demonstriert Ihnen die Wirkung aller vier Routinen. Die Maschinenprogramme müssen dazu unter dem Namen „TEXTLOESCH1.OBJ“ bis „TEXTLOESCH4.OBJ“ auf der Diskette vorliegen.

TL.DEMO

```
5   REM  TEXTBILDSCHIRM-LOESCHER
6   A = 1: TEXT : HOME
7   A$ =  STR$ (A)
8   PRINT : PRINT CHR$ (4)"BLOAD TEXTLOESCH"A$".OBJ"
10  FOR I = 1 TO 919: PRINT "*";: NEXT
15  GET A$
20  CALL 768
25  IF A < 4 THEN A = A + 1: GOTO 7
30  END
```

1.4 Ausdrucksvermögen

Wie wir einen Text aus einem Assembler-Programm auf den Bildschirm ausgeben können, haben wir in Lektion 20 des ersten Bandes bereits gesehen. Wir benutzten dazu die Monitor-Routine COUT ($FDED), die das Zeichen im Akku an der aktuellen Cursorposition unter Beachtung der Fenstergrenzen setzt.

COUT $FDED
Gibt das Zeichen im Akku auf das aktuelle Ausgabegerät (CSW-Vektor siehe Seite 86) aus. Beim Bildschirm werden die Fenstergrenzen beachtet. Die drei Register Akku, X-Reg und Y-Reg werden zwischengespeichert und so gerettet.

Eingabe:
CSWL/H Zeiger auf Ausgabegerät (40Z/Z-Bildschirm = $FDF0)
Akku = auszugebendes Zeichen

Ausgabe: Cursor-Position wird weitergesetzt

PRINT1

```
                        1   ;******************
                        2   ; Print Nummer 1 *
                        3   ;******************
                        4   ;
                        5             ORG $300
                        6   ;
                        7   DOSWRM    EQU $03D0
                        8   COUT      EQU $FDED
                        9   ;
0300: A0 00             10  PRINT1    LDY #00         ;Laufzähler
0302: B9 10 03          11  PR1       LDA TEXT,Y      ;Stringbyte
0305: F0 06             12            BEQ ENDE
0307: 20 ED FD          13            JSR COUT        ;ausgeben
030A: C8                14            INY             ;nächstes Zeichen
030B: D0 F5             15            BNE PR1         ;fast immer
030D: 4C D0 03          16  ENDE      JMP DOSWRM
                        17  ;
0310: C2 E5 E9          18  TEXT      ASC "Beispielsatz"
031C: 00                19            HEX 00
```

Dieses Verfahren ist angebracht, wenn in einem Programm nur wenige Texte ausgegeben werden müssen. Wollen Sie dieses Programm in ein größeres einbauen, so ersetzen Sie das „JMP DOSWRM' durch „RTS".

Kommen viele Texte vor, dann ist es eine Platzverschwendung, wenn für jede Ausgabe eine eigene Druckroutine geschrieben werden muß. Hier ist ein Unterprogramm notwendig, mit dem alle Texte ausgegeben werden können, wenn ihm deren Startadresse und Länge mitgeteilt wird. Das klassische Beispiel ist die PRINT-Routine von Andy Hertzfeld:

PRINT2

```
                        1   ;******************
                        2   ; Print Nummer 2  *
                        3   ;******************
                        4   ;
                        5   ; nach Andy Hertzfeld
                        6   ;
                        7             ORG $300
                        8   ;
                        9   PTR       EQU $0006
                        10  DOSWRM    EQU $03D0
                        11  COUT      EQU $FDED
                        12  ;
0300: 20 10 03          13  DEMO      JSR PRINT2
0303: D4 E5 F8          14            ASC "Textprobe"
030C: 00                15            HEX 00
030D: 4C D0 03          16            JMP DOSWRM
                        17  ;
0310: 68                18  PRINT2    PLA             ;hole Return-
0311: 85 06             19            STA PTR         ;adresse vom
0313: 68                20            PLA             ;Stack
```

```
0314: 85 07      21              STA PTR+1
0316: A0 01      22              LDY #$01        ; +1
0318: B1 06      23    PR2       LDA (PTR),Y     ;String
031A: F0 06      24              BEQ ENDE
031C: 20 ED FD   25              JSR COUT        ;ausgeben
031F: C8         26              INY
0320: D0 F6      27              BNE PR2         ;fast immer
                 28    ;
0322: 18         29    ENDE      CLC             ;Stringlänge
0323: 98         30              TYA             ;addieren und
0324: 65 06      31              ADC PTR
0326: 85 06      32              STA PTR         ;(merken)
0328: A5 07      33              LDA PTR+1
032A: 69 00      34              ADC #$00
032C: 48         35              PHA             ;auf den Stack
032D: A5 06      36              LDA PTR
032F: 48         37              PHA             ;zurückbringen
0330: 60         38              RTS             ;anspringen
```

Ein Aufruf hat immer die Form

```
JSR PRINT2
ASC "......"   ;Ihr Text
HEX 00
```

Die Druckroutine PRINT2 kann an beliebiger Stelle stehen. Ihrem Aufruf durch „JSR" folgt der Text, der nach dem Pseudo-Opcode „ASC" eingegeben wird. Für inversen oder blinkenden Text können Sie auch „INV" bzw. „FLS" verwenden. Das Textende ist mit „$00" gekennzeichnet. Dazu müssen Sie „HEX 00" eingeben. Ein „BRK" führt aber auch zum selben Ergebnis.

Wie arbeitet nun dieses Programm? Bei dem Befehl „JSR PRINT2" wird die Rücksprungadresse − 1 auf den Stack geschoben. In unserem Beispiel ist das $0302. Diese Adresse wird durch die zwei „PLA"-Befehle wieder vom Stack zurückgeholt und auf der Null-Seite (Zero-Page) in „PTR" und „PTR+1" zwischengespeichert. (PTR) zeigt dann nach $0302. Unser Text beginnt durch die Art des Aufrufs *immer* an der folgenden Adresse, hier bei $0303. Auf diese Weise „weiß" die Druckroutine, wo der Text steht. Indirekt indiziert können Sie ihn nun lesen. Wegen des um 1 versetzten Beginns müssen Sie mit dem Wert 1 im Y-Register anfangen. Die sich anschließende Schleife arbeitet im Prinzip wie in PRINT1. Wieder zählt das Y-Register die Zahl der ausgegebenen Zeichen mit. Aus diesem Grunde kann der Text maximal 254 Zeichen lang sein. Die „00" dient als Abbruchkriterium. Die Schleife wird über „BEQ ENDE" verlassen. Die Druckroutine ist damit noch nicht ganz zu Ende. Durch die zwei „PLA" befindet sich keine gültige Rücksprungadresse mehr auf dem Stack. Der Rücksprung darf nicht direkt hinter den Aufruf „JSR PRINT2" erfolgen, da hier ja

Text steht und kein ausführbarer Maschinencode. Wir müssen also hinter das „HEX 00" springen. Dazu muß die Adresse von „HEX 00" auf den Stack gebracht werden, denn sie entspricht der Rücksprungadresse – 1.
Woher bekommt das Programm diese Adresse? In „PTR" und „PTR+1" steht noch die alte Rücksprungadresse – 1. Das Y-Register hat die Zahl der Zeichen vom Textbeginn bis zum „HEX 00" mitgezählt. Wenn wir das Y-Register zu (PTR) addieren, ergibt sich die Adresse des „HEX 00". Davon muß zunächst das Hi- und dann das Lo-Byte auf den Stack gebracht werden, bevor uns das „RTS" an die richtige Stelle (Adresse vom Stack +1) ins Hauptprogramm zurückführt.
Bei diesem Programm ist es wieder lohnend, sich die einzelnen Schritte im IDUS anzusehen. Lassen Sie sich in der freien Speicheranzeige die Adressen $0006 und $0007 anzeigen, und setzen Sie den Wert für „Schnellgang" auf $F800.

Unser bisheriges Druckprogramm hat den Nachteil, daß der Text in der gerade aktiven Zeile ausgegeben wird, ohne daß wir einen Einfluß darauf hätten. Wollen wir selbst den Ort bestimmen, müssen wir dem Aufruf noch ein paar Zeilen voranstellen. Zunächst wird der horizontale Tabulatorwert ($00 - $17) nach CH ($0024) geschrieben. Dann rufen wir mit dem vertikalen Tabulatorwert im Akkumulator die Routine TABV ($FB5B) auf, die uns CV und BASL/H richtig setzt, so daß der Text an der gewünschten Position erscheint.

TABV $FB5B
Vertikale Tabulierung unter Berücksichtigung des linken Fensterrandes.

Eingabe:
Akku = gewünschter VTAB-Wert ($00-$17)

Ausgabe:
CV = VTAB-Wert
BASL/H = Basisadresse der Zeile

PRINT3

```
                    1   ;******************
                    2   ; PRINT Nummer 3  *
                    3   ;******************
                    4   ;
                    5             ORG $300
                    6   ;
                    7   PTR       EQU $0006
                    8   CH        EQU $0024
                    9   DOSWRM    EQU $03D0
                   10   TABV      EQU $FB5B
                   11   COUT      EQU $FDED
                   12   ;
0300: A9 05        13   DEMO2     LDA #5          ;Cursor horiz.
0302: 85 24        14             STA CH
0304: A9 05        15             LDA #5          ;Cursor vert.
0306: 20 5B FB     16             JSR TABV
0309: 20 2B 03     17             JSR PRINT3
030C: D4 E5 F8     18             ASC "Textprobe"
0315: 8D 8D A0     19             HEX 8D8DA0A0
0319: 09 0E 20     20             INV "IN NEUER ZEILE"
0327: 00           21             HEX 00
0328: 4C D0 03     22             JMP DOSWRM
                   23   ;
032B: 68           24   PRINT3    PLA             ;hole Return-
032C: 85 06        25             STA PTR         ;adresse vom
032E: 68           26             PLA             ;Stack
032F: 85 07        27             STA PTR+1
0331: A0 01        28             LDY #$01        ; +1
0333: B1 06        29   PR3       LDA (PTR),Y     ;String
0335: F0 06        30             BEQ ENDE
0337: 20 ED FD     31             JSR COUT        ;ausgeben
033A: C8           32             INY
033B: D0 F6        33             BNE PR3         ;fast immer
                   34   ;
033D: 18           35   ENDE      CLC             ;Stringlänge
033E: 98           36             TYA             ;addieren und
033F: 65 06        37             ADC PTR
0341: 85 06        38             STA PTR         ;(merken)
0343: A5 07        39             LDA PTR+1
0345: 69 00        40             ADC #$00
0347: 48           41             PHA             ;auf den Stack
0348: A5 06        42             LDA PTR
034A: 48           43             PHA             ;zurückbringen
034B: 60           44             RTS             ;anspringen
```

„PRINT3" zeigt Ihnen, wie Sie innerhalb des Textes noch Anweisungen unterbringen können, die Sie nicht über „ASC" usw. einfügen. „8D" ist der Code für <RETURN>, „$A0" steht für ein Leerzeichen. Die zweite Zeile wird also zwei Zeilen unter der ersten stehen und um 2 Zeichen eingerückt sein. Sie können alle HEX-Werte von $01 bis $FF auf diese Weise in die Ausgabe einfügen.

„$87" entspricht z.B. Ctrl-G, was zu einem „Piep" im Lautsprecher führt. Probieren Sie es einmal aus.

Die bisherigen Routinen waren durch die indizierte indirekte Adressierung in der Länge des Ausgabetextes begrenzt. Mit „PRINT4" können wir dagegen beliebig lange Texte ausgeben, da hier die Zeiger auf der Null-Seite weitergesetzt werden und Y konstant bleibt. Auch für die Cursor-Steuerung enthält dieses Programm eine Neuerung. Die ersten beiden Bytes nach dem Aufruf der Druckroutine („JSR PRINT4") werden als horizontaler bzw. vertikaler Tabulatorwert aufgefaßt. Erst danach folgt der eigentliche Text. Wenn Sie viel tabulieren müssen, spart dies gegenüber „PRINT3" noch einmal Speicherplatz.

PRINT4

```
                        1   ;******************
                        2   ; Print Nummer 4  *
                        3   ;******************
                        4   ;
                        5             ORG $300
                        6   ;
                        7   PTR       EQU $0006
                        8   CH        EQU $0024
                        9   DOSWRM    EQU $03D0
                       10   TABV      EQU $FB5B
                       11   COUT      EQU $FDED
                       12   ;
0300: 20 11 03         13   DEM03     JSR PRINT4
0303: 0A 05            14             HEX 0A05        ;Cur. horz., vert.
0305: C2 E5 E9         15             ASC "Beispiel"
030D: 00               16             HEX 00
030E: 4C D0 03         17             JMP DOSWRM
                       18   ;
0311: 68               19   PRINT4    PLA             ;Returnadresse
0312: 85 06            20             STA PTR         ;vom Stack
0314: 68               21             PLA             ;ziehen und
0315: 85 07            22             STA PTR+1       ;merken
0317: A0 00            23             LDY #00         ;initialisieren
0319: 20 35 03         24             JSR INKREM      ;nächst. Z.
031C: 85 24            25             STA CH          ;Cursor horz.
031E: 20 35 03         26             JSR INKREM      ;nächst. Z.
0321: 20 5B FB         27             JSR TABV        ;Zeiger setzen
0324: 20 35 03         28   PR4       JSR INKREM      ;nächst. Z.
0327: F0 06            29             BEQ ENDE
0329: 20 ED FD         30             JSR COUT        ;String ausgeben
032C: 4C 24 03         31             JMP PR4         ;immer
                       32   ;
032F: 20 35 03         33   ENDE      JSR INKREM      ;hinter Text
0332: 6C 06 00         34             JMP (PTR)       ;zurückspringen
                       35   ;
0335: E6 06            36   INKREM    INC PTR         ;Zeiger wei-
0337: D0 02            37             BNE LADEN       ;terschieben
0339: E6 07            38             INC PTR+1
033B: B1 06            39   LADEN     LDA (PTR),Y     ;Z. laden
033D: 60               40             RTS
```

Schauen Sie sich auch dieses Beispiel im IDUS an, besonders den Rücksprung in Zeile 33/34.

Die verschiedenen Möglichkeiten einer Druckroutine sind noch längst nicht ausgereizt. „PRINT5" zeigt eine weitere Variation sowohl in der Cursorsteuerung, die hier über den Akkumulator und das X-Register erfolgt, als auch für das Weitersetzen des Zeigers (PTR). Die dritte Änderung liegt in der Kennzeichnung des Textendes. Bisher brauchten wir mit „$00" ein Extrabyte. Jetzt benutzen wir das Bit 7, um das letzte Zeichen zu finden. „DCI" invertiert im letzten Zeichen der Zeichenkette das Vorzeichen. In diesem Beispiel ist Bit 7 bis auf das letzte Zeichen immer gesetzt. Das Vorzeichen jedes ASCII-Wertes wird gerettet (Zeile 42) und nach der Ausgabe wieder zurückgeholt (Zeile 45). War Bit 7 gesetzt, geht der Text noch weiter, war Bit 7 gelöscht – der Wert also positiv –, ist das Ende erreicht. Da wir bei dieser Technik nicht die Pseudo-Opcodes „INV" und „FLS" verwenden dürfen, benutzen wir eine andere Möglichkeit, zusammen mit COUT die Attribute zu bestimmen. Die Monitor-Routine SETINV ($FE80) sorgt für inverse Textausgabe, SETNORM ($FE84) schaltet auf normale Ausgabe. Ein FLASH müssen Sie selber schreiben durch

```
LDA #$7F
STA INVFLG (= $0032).
```

SETNORM $FE84
Schaltet auf normale Bildschirmdarstellung bei Textausgabe über COUT.

Eingabe: –

Ausgabe: INVFLG = $FF, Y-Reg = $FF

SETINV $FE80
Schaltet auf inverse Bildschirmdarstellung bei Textausgabe über COUT.

Eingabe: –

Ausgabe: INVFLG = $3F, Y-Reg = $3F

PRINT5

```
                        1   ;******************
                        2   ; PRINT Nummer 5  *
                        3   ;******************
                        4   ;
                        5           ORG $300
                        6   ;
                        7   PTR     EQU $0006
                        8   CH      EQU $0024
                        9   DOSWRM  EQU $03D0
                       10   TABV    EQU $FB5B
                       11   COUT    EQU $FDED
                       12   SETINV  EQU $FE80
                       13   SETNORM EQU $FE84
                       14   ;
0300: A9 05            15   DEM04   LDA #5          ;Cursor vert.
0302: A2 0A            16           LDX #10         ;Cursor horz.
0304: 20 41 03         17           JSR PRINT5
0307: CE CF D2         18           DCI "NORMAL UND"
0311: A9 07            19           LDA #7
0313: A2 0A            20           LDX #10
0315: 20 3E 03         21           JSR PRINT5I
0318: CA C5 D4         22           DCI "JETZT INVERS"
0324: 20 84 FE         23           JSR SETNORM
0327: A9 09            24           LDA #9
0329: A2 0A            25           LDX #10
032B: 20 41 03         26           JSR PRINT5
032E: D7 C9 C5         27           DCI "WIEDER NORMAL"
033B: 4C D0 03         28           JMP DOSWRM
                       29   ;
033E: 20 80 FE         30   PRINT5I  JSR SETINV     ;Inversmaske
0341: 86 24            31   PRINT5   STX CH         ;Cursor horz.
0343: 20 5B FB         32           JSR TABV        ;Cursor vert.
0346: 68               33           PLA             ;Returnadresse
0347: 85 06            34           STA PTR         ;vom Stack
0349: 68               35           PLA             ;und speichern
034A: 85 07            36           STA PTR+1
034C: A0 00            37           LDY #00         ;initialsieren
034E: E6 06            38   PR5     INC PTR         ;nächstes Z.
0350: D0 02            39           BNE PR5A        ;Übertrag?
0352: E6 07            40           INC PTR+1       ;JA
0354: B1 06            41   PR5A    LDA (PTR),Y     ;Zeichen laden
0356: 48               42           PHA             ;Vorzeichen merken
0357: 09 80            43           ORA #$80        ;Bit 7 setzen
0359: 20 ED FD         44           JSR COUT        ;ausgeben
035C: 68               45           PLA             ;Vorzeichen holen
035D: 30 EF            46           BMI PR5         ;Positiv als Ende
035F: A5 07            47           LDA PTR+1       ;Rücksprungadresse
0361: 48               48           PHA             ;auf den
0362: A5 06            49           LDA PTR         ;Stack schieben
0364: 48               50           PHA
0365: 60               51           RTS             ;anspringen
```

Wenn Sie den Applesoft-Interpreter mitbenutzen wollen, können Sie eine ganz kurze Druckroutine schreiben. STROUT ($DB3A) gibt einen Text mit bis zu 255 Zeichen über COUT aus. Der Text kann allerdings nicht direkt hinter dem Aufruf mit „JSR STROUT" liegen, da STROUT die Rücksprungadresse nicht weitersetzt. Sie benötigen also wie bei PRINT1 einen separaten Platz für den Text. Die Adresse des ersten Bytes muß im Akkumulator (Lo) und Y-Register (Hi) übergeben werden. Alle Zeichen müssen mit gelöschtem Bit 7 vorliegen. Das Textende muß durch „$00" oder „$22" gekennzeichnet sein. Das bedeutet, daß Sie Anführungszeichen oben (") nicht mitdrucken können!!

STROUT $DB3A
Gibt eine Zeichenkette aus, die mit Bit 7 = 0 vorliegt und mit $00 oder $22 endet.

Eingabe:
Akku = Lo-Byte vom Zeichenkettenstart
Y-Reg = Hi-Byte vom Zeichenkettenstart

Ausgabe:
Akku = letztes ausgegebenes Zeichen
Y-Reg = Anzahl der Zeichen + 1
TXTPTR zeigt auf Zeichen hinter dem String

PRINT6

```
                      1    ;******************
                      2    ; PRINT Nummer 6  *
                      3    ;******************
                      4    ;
                      5    ; benutzt Applesoft
                      6    ;
                      7              ORG $300
                      8    ;
                      9    DOSWRM    EQU $03D0
                      10   STROUT    EQU $DB3A
                      11   INIT      EQU $FB2F
                      12   HOME      EQU $FC58
                      13   ;
0300: 20 2F FB        14   DEMO5     JSR INIT       ;Textbildschirm
0303: 20 58 FC        15             JSR HOME       ;löschen
0306: A9 10           16             LDA #<TEXT     ;Lo-Byte
0308: A0 03           17             LDY #>TEXT     ;Hi-Byte
030A: 20 3A DB        18             JSR STROUT     ;String ausgeben
030D: 4C D0 03        19             JMP DOSWRM
                      20   ;
```

```
0310: 54 65 78  21   TEXT      ASC 'Text mit Applesoft'
0322: 00        22             HEX 00          ;Ende-Kennung
```

Die Textausgabe über STROUT hat noch eine weitere Eigenschaft, die sie von allen anderen Routinen unterscheidet: Sie können die Geschwindigkeit steuern. Je nach Inhalt von SPEED ($00F1) werden zwischen den einzelnen Zeichen Verzögerungsschleifen durchlaufen. Ein Wert von $01 ergibt die maximale Geschwindigkeit, die bis $FF immer weiter herabgesetzt wird. Mit $00 erreichen Sie die größte Verzögerung.

PRINT7

```
                     1    ;******************
                     2    ; PRINT Nummer 7  *
                     3    ;******************
                     4    ;
                     5    ; benutzt Applesoft
                     6    ;
                     7              ORG $300
                     8    ;
                     9    CH        EQU $0024
                     10   SPEED     EQU $00F1
                     11   DOSWRM    EQU $03D0
                     12   STROUT    EQU $DB3A
                     13   INIT      EQU $FB2F
                     14   TABV      EQU $FB5B
                     15   HOME      EQU $FC58
                     16   ;
0300: 20 2F FB       17   DEM06     JSR INIT        ;Textbildschirm
0303: 20 58 FC       18             JSR HOME        ;löschen
0306: A9 80          19             LDA #$80        ;verzögern
0308: 85 F1          20             STA SPEED
030A: A9 05          21             LDA #5
030C: 85 24          22             STA CH
030E: A9 07          23             LDA #7
0310: 20 5B FB       24             JSR TABV
0313: A9 21          25             LDA #<TEXT      ;Lo-Byte
0315: A0 03          26             LDY #>TEXT      ;Hi-Byte
0317: 20 3A DB       27             JSR STROUT      ;String ausgeben
031A: A9 01          28             LDA #01         ;zurücksetzen
031C: 85 F1          29             STA SPEED
031E: 4C D0 03       30             JMP DOSWRM
                     31   ;
0321: 54 65 78       32   TEXT      ASC 'Text mit Applesoft'
0333: 00             33             HEX 00          ;Ende-Kennung
```

Jede der hier vorgestellten Routinen hat ihre Vor- und Nachteile. Welche Sie im konkreten Fall anwenden, hängt zum Teil von den Anforderungen des Programms ab, ist aber auch Ihrer freien Entscheidung (= Geschmack?) unterworfen. Ich persönlich benutze am häufigsten „PRINT2".

1.5 80 Zeichen: Schalten und Verwalten

Professionelle Programme verlangen in der Regel nach einer Textausgabe mit 80 Zeichen pro Zeile. Beim alten Apple II+ ist dies nur mit einer Zusatzkarte zu erreichen. Am verbreitetsten ist das VIDEX-System, das einen eigenen Bildschirmspeicher besitzt. Nähere Informationen zum Aufbau des VIDEX-Bildschirms finden Sie im Peeker Nr. 1/2 von 1985 ab Seite 42. Der Apple IIe benötigt eine andere Zusatzkarte, die im IIc bereits fest eingebaut ist. Die Hälfte der Daten findet sich dabei an den selben Speicherstellen wie bei der 40-Zeichen-Darstellung: von $0400 bis $07FF. Die andere Hälfte befindet sich auf den selben Adressen des Zusatzspeichers (AUX-Memory) der Erweiterungskarte. Da beide Speicherbereiche den selben Adressraum belegen, muß zwischen ihnen hin- und hergeschaltet werden. Dies wird von der Anzeigeelektronik automatisch vorgenommen. Die Spalten mit ungerader Nummer liegen im Hauptspeicher (Main), die Spalten mit gerader Nummer im Zusatzspeicher (AUX).

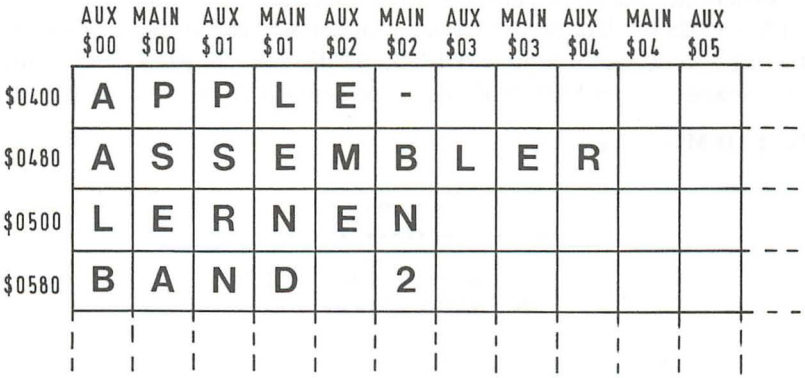


Abb. 2: Adressierung der 80Z/Z Textseite

Um Daten direkt auf den 80-Zeichen-Schirm zu speichern oder zu lesen, müssen Sie zunächst den Softswitch 80STOREON ($C001) betätigen, indem Sie irgendeinen beliebigen Wert dorthin **speichern** (STA $C001). Wenn 80STORE auf diese Weise angeschaltet wurde, wird mit den Softswitches LOWSCR ($C054) und HISCR ($C055) nicht mehr zwischen den Textseiten 1 und 2 hin- und

hergeschaltet, sondern zwischen dem Haupt- und dem Erweiterungsspeicher der 80-Zeichenkarte.

```
ungerade Spalten bearbeiten      gerade Spalten bearbeiten
   LDA #$00                         LDA #$00
   STA 80STOREON ($C001)            STA 80STOREON($C001)
   STA LOWSCR    ($C054)            STA HISCR    ($C055)
```

80STORE wird wieder abgeschaltet durch das Schreiben nach 80STOREOFF ($C000).

Achtung: Diese Adresse ist identisch mit der Tastaturadresse. Wenn Sie aus dieser Adresse lesen (z.B. LDA, BIT), erhalten Sie die Tastaturinformation. Wenn Sie in diese Adresse schreiben (z.B. STA), schalten Sie 80STORE ab.

Wenn Sie nur einfach Texte über COUT auf den 80-Zeichen-Schirm ausgeben wollen, brauchen Sie sich um die Softswitches nicht zu kümmern. Sie müssen lediglich das Zusatz-ROM im Adressbereich von $C300-$C3FF einschalten durch ein „JSR $C300". COUT funktioniert dann automatisch auch mit der 80-Zeichenkarte, und Sie können unsere Druckroutinen PRINT1 bis PRINT7 benutzen. Ein Schönheitsfehler ist allerdings dabei: der horizontale Tabulator funktioniert nicht richtig, da die 80-Zeichenkarte den horizontalen Cursorwert nicht bei $0025 sondern bei $057B in einem der Scratchpad-Bytes verwaltet. Erfreulicherweise tut auch die VIDEX-Karte dies, so daß das folgende Programm sowohl mit der VIDEX-Karte als auch mit der Apple-Karte läuft.

PRINT DEMO 80Z/Z

```
1    ;*********************
2    ; PRINT DEMO 80Z/Z  *
3    ;*********************
4    ;
5    ; Apple IIe/c oder VIDEX
6    ; 80 Zeichenkarte in Slot 3
7    ;
8              ORG $300
9    ;
10   PTR       EQU $0006
11   HORZ      EQU $0008
12   DOSWRM    EQU $03D0
13   CH        EQU $057B
14   BASICINT  EQU $C300
15   TABV      EQU $FB5B
16   COUT      EQU $FDED
17   SETINV    EQU $FE80
18   SETNORM   EQU $FE84
19   ;
```

```
0300: A9 00      20  DEMO7     LDA #00         ;initialisieren
0302: 85 08      21              STA HORZ
0304: 20 00 C3   22              JSR BASICINT    ;80Z/Z ein
0307: 20 18 03   23  DM          JSR DM2
030A: 18         24              CLC
030B: A5 08      25              LDA HORZ
030D: 69 0D      26              ADC #13
030F: 85 08      27              STA HORZ
0311: C9 42      28              CMP #66
0313: 90 F2      29              BLT DM
0315: 4C D0 03   30              JMP DOSWRM
                 31  ;
0318: A9 05      32  DM2         LDA #5          ;Cursor vert.
031A: A6 08      33              LDX HORZ        ;Cursor horz.
031C: 20 57 03   34              JSR PRINT8
031F: CE CF D2   35              DCI "NORMAL UND"
0329: A9 07      36              LDA #7
032B: A6 08      37              LDX HORZ
032D: 20 54 03   38              JSR PRINT8I
0330: CA C5 D4   39              DCI "JETZT INVERS"
033C: 20 84 FE   40              JSR SETNORM
033F: A9 09      41              LDA #9
0341: A6 08      42              LDX HORZ
0343: 20 57 03   43              JSR PRINT8
0346: D7 C9 C5   44              DCI "WIEDER NORMAL"
0353: 60         45              RTS
                 46  ;
0354: 20 80 FE   47  PRINT8I     JSR SETINV      ;Inversmaske
0357: 8E 7B 05   48  PRINT8      STX CH          ;Cursor horz.
035A: 20 5B FB   49              JSR TABV        ;Cursor vert.
035D: 68         50              PLA             ;Returnadresse
035E: 85 06      51              STA PTR         ;vom Stack
0360: 68         52              PLA             ;und speichern
0361: 85 07      53              STA PTR+1
0363: A0 00      54              LDY #00         ;initialsieren
0365: E6 06      55  PR8         INC PTR         ;nächstes Z.
0367: D0 02      56              BNE PR8A        ;Übertrag?
0369: E6 07      57              INC PTR+1       ;JA
036B: B1 06      58  PR8A        LDA (PTR),Y     ;Zeichen laden
036D: 48         59              PHA             ;Vorzeichen merken
036E: 09 80      60              ORA #$80        ;Bit 7 setzen
0370: 20 ED FD   61              JSR COUT        ;ausgeben
0373: 68         62              PLA             ;Vorzeichen holen
0374: 30 EF      63              BMI PR8         ;Positiv als Ende
0376: A5 07      64              LDA PTR+1       ;Rücksprungadresse
0378: 48         65              PHA             ;auf den
0379: A5 06      66              LDA PTR         ;Stack schieben
037B: 48         67              PHA
037C: 60         68              RTS             ;anspringen
```

2. Blockmalereien

Neben der Textdarstellung besitzt der Apple die Fähigkeit, Grafiken in niedriger und hoher Auflösung anzuzeigen. Die niedrigauflösende Grafik (LORES) erfreut sich nur mäßiger Beliebtheit, obwohl sie immerhin über 16 Farben verfügt und einfach zu programmieren ist. Wenn Sie farbige Rechtecke darstellen müssen, gibt es zu LORES keine Alternative. Der Nachteil von LORES ist seine grobe blockartige Struktur, da die Anzeige aus 1920 kleinen Rechtecken besteht, die in 48 Reihen zu je 40 Spalten angeordnet sind.

2.1 Aus 1 mach 2, das ist das LORES 1x1

Die LORES-Grafik belegt den selben Speicherplatz wie die Textseiten. Es gibt ebenfalls zwei anzeigbare LORES-Seiten: LORES1 beginnt bei $0400 und endet mit $07FF, während LORES2 den Speicher von $0800 bis $0BFF beansprucht. Zu jedem Byte gehören 2 übereinander liegende Rechtecke, deren Farbe unabhängig voneinander wählbar ist. Die Organisation des LORES-Bildschirms entspricht dem der Textseite, wenn Sie sich vorstellen, daß jede Textzeile horizontal in zwei Grafikzeilen gespalten ist. Zu Textzeile 0 gehören die Grafikzeilen 0 und 1, zu Textzeile 1 gehören die Grafikzeilen 2 und 3 usw.
Jedes Speicherbyte besteht aus 2 Nibbles. In der hexadezimalen Schreibweise sind sie mit den beiden Ziffern der HEX-Zahl identisch. $F6 besteht aus dem Nibble $F und dem Nibble $6. Das linke Nibble bestimmt die Farbe der ungeradzahligen Grafikzeilen (= unten), das rechte Nibble ist für die geradzahligen Zeilen (= oben) da. Ein Nibble kann die 16 Werte von $0 bis $F annehmen. Ihnen entsprechen die 16 darstellbaren Farben.

LORES Farbtabelle

Wert	Farbe	Wert	Farbe
$0	Schwarz	$8	Braun
$1	Fuchsinrot	$9	Orange
$2	Dunkelblau	$A	Grau 2
$3	Purpur	$B	Rosa
$4	Dunkelgrün	$C	Hellgrün
$5	Grau 1	$D	Gelb
$6	Blau	$E	Aquamarin
$7	Hellblau	$F	Weiß

Die Farbbezeichnungen sind so von der Firma Apple angegeben worden. Die tatsächlichen Farben auf Ihrem Monitor können davon abweichen.
Das Byte $6D würde demnach ein gelbes Rechteck über einem blauen zeigen.

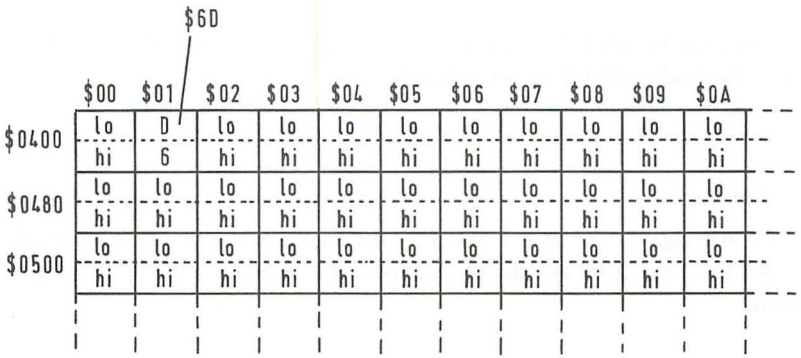


Abb. 3: Adressierung der LORES-Seite 1

2.2 Switch softly

Bereits beim Textbildschirm haben wir kennengelernt, daß die Anzeige des Apple durch einige Speicherstellen gesteuert wird, die „Softswitches" genannt werden. Wenn wir die 80-Zeichen-Karte des IIe und IIc einmal außer acht lassen, gibt es folgende Möglichkeiten:

Schalter	Funktion
$C050 TXTCLR	Schaltet Grafik ein
$C051 TXTSET	Schaltet Text ein
$C052 MIXCLR	Nur Text oder Grafik
$C053 MIXSET	Grafik mit 4 Zeilen Text *
$C054 LOWSCR	Seite 1 von Grafik oder Text
$C055 HISCR	Seite 2 von Grafik oder Text
$C056 LORES	niedrigauflösende Grafik *
$C057 HIRES	hochauflösende Grafik *

* Diese Zustände sind nur sichtbar, wenn TXTCLR betätigt wird.

Sie können die Schalter stellen, indem Sie entweder aus der Speicherstelle lesen oder in diese schreiben. Durch die Befehlsfolge

```
LDA TXTCLR
LDA MIXCLR
LDA LOWSCR
LDA LORES
```

wird die Anzeige des gesamten LORES-Grafikschirms der Seite 1 ausgewählt. Es kommt dabei nicht auf die Reihenfolge an, in der die Schalter betätigt werden!

2.3 Mehr als ein Regenbogen

Die von Applesoft benutzten Routinen für das Zeichen von LORES-Grafik sind Bestandteil des Monitors. Wir können sie von einem Assembler-Programm aus sehr einfach mitbenutzen. Sie funktionieren allerdings nur für die LORES-Seite 1.
Das folgende Demonstrationsprogramm setzt zunächst den LORES-Grafikschirm mit 4 Textzeilen am unteren Rand durch den Aufruf von SETGR ($FB40). Dies funktioniert allerdings nur, wenn vorher der Textbildschirm

angeschaltet war. Wollen Sie auch von HGR nach LORES umschalten, müssen Sie die Applesoft-Routine GR ($F390) benutzen.

SETGR $FB40
Schaltet um auf Grafik und gemischte Darstellung. Das Textfenster wird geöffnet, der obere Teil gelöscht, wenn es sich um LORES handelt.

Eingabe: –

Ausgabe:
Y-Reg = $FF, BASL/H zeigt auf Zeile $17
COLOR = $00 gesetzt

GR $F390
Schaltet auf LORES-Grafik und gemischte Darstellung. Textfenster geöffnet, Cursor in der letzten Zeile, LORES-Farbe Schwarz aktiv.

Eingabe: –

Ausgabe:
Y-Reg = $FF, BASL/H zeigt auf Zeile $17
COLOR = $00

Als nächstes werden 16 horizontale Linien in allen Farben gezeichnet. Zum Setzen einer Farbe wird deren Nummer in den Akkumulator geladen und dann SETCOL ($F864) aufgerufen.
Horizontale Linien werden entsprechend dem BASIC-Befehl HLIN XL,XR AT Y durch HLINE ($F819) gezogen, wobei die Koordinaten im Y-Register (XL), der Zero-Page-Adresse H2 ($002C) (XR) und im Akkumulator (Y) übergeben werden.

SETCOL $F864
Setzt die LORES-Farbe

Eingabe: Akku = Farbnummer ($0...$F)

Ausgabe: COLOR ($0030) gesetzt

HLINE $F819
Zeichnet horizontale Linie von XLinks nach XRechts in der Zeile Y.

Eingabe:
COLOR gesetzt, Y-Reg = XL, H2 ($003C) = XR, Akku = Y

Ausgabe: –

Um den Bildschirm mit vertikalen Linien zu füllen, benutzt das Demo-Programm die Routine VLINE ($F828). Dies geschieht in einer Schleife, der das Y-Register als Laufvariable dient. Da die Farbwerte nur zwischen $00 und $0F liegen dürfen, wird mit einem „AND $0F" das obere Nibble ausgeblendet.

VLINE $F828
Zeichnet vertikale Linie von YOben bis YUnten in der Spalte X.

Eingabe:
COLOR gesetzt, Akku = YO, V2 ($002D) = YU, Y-Reg = X

Ausgabe: –

Darunter kann dann der Benutzer eine Doppellinie eingeben, indem jeder Tastendruck als ASCII-Zeichen direkt in den LORES-Speicher geschrieben und damit sichtbar wird. Die Routine CLRTOP ($F836) löscht die oberen 40 LORES-Zeilen, läßt aber den Text intakt. Mit CLRSCR ($F832) können Sie den ganzen LORES-Schirm löschen, wenn Sie den Softswitch $C052 vorher betätigen. Bei der gemischten Darstellung füllt sich das Textfenster mit inversen @.

CLRTOP $F836
Löscht die oberen 40 Zeilen der LORES-Grafik 1.

Eingabe: –

Ausgabe: Y-Reg = $FF, Akku = $27, COLOR = $00

CLRSCR $F832
Löscht alle 48 Zeilen der LORES-Grafik 1.

Eingabe: –

Ausgabe: Y-Reg = $FF, Akku = $2F, COLOR = $00

CLRSC2 $F838
Löscht Y Zeilen der LORES-Grafik 1.

Eingabe:
Y-Reg = Zahl der zu löschenden Zeilen (von oben).

Ausgabe: Y-Reg = $FF, Akku = Zeilenzahl, COLOR = $00

Den Abschluß des Demoprogramms bildet eine kleine Blockgrafik, die Punkt für Punkt gesetzt wird. Die Koordinaten liegen in einer Tabelle und werden mit PLOT ($F800) umgesetzt.

PLOT $F800
Zeichnet einen Punkt X,Y in der Farbe COLOR.

Eingabe:
COLOR gesetzt, Akku = Y, Y-Reg = X

Ausgabe: –

LORES

```
                        1   ;*******************************
                        2   ;  Lo-Res  Beispielprogramm   *
                        3   ; (C) 1986  Dr. Jürgen Kehrel  *
                        4   ;*******************************
                        5   ;
                        6   ;
                        7   H2          EQU $002C
                        8   V2          EQU $002D
                        9   ;
                        10  KBD         EQU $C000
                        11  STROBE      EQU $C010
                        12  ;
                        13  PLOT        EQU $F800
                        14  HLINE       EQU $F819
                        15  VLINE       EQU $F828
                        16  CLRSCR      EQU $F832
                        17  CLRTOP      EQU $F836
                        18  SETCOL      EQU $F864
                        19  SETGR       EQU $FB40
                        20  ;
                        21  ;
                        22              ORG $0803
                        23  ;
0803: 20 40 FB          24  START       JSR SETGR       ;Grafik ein
                        25  ;
                        26  ; 16 horz. Linien in allen Farben
                        27  ;
0806: A2 0F             28              LDX #$0F        ;Beginn „Weiß"
0808: 8A                29  LOOP        TXA
0809: 20 64 F8          30              JSR SETCOL      ;Farbe
080C: A0 00             31              LDY #$00        ;X-Links
080E: A9 27             32              LDA #$27        ;X-Rechts
0810: 85 2C             33              STA H2
0812: 8A                34              TXA             ;Y-Wert
0813: 20 19 F8          35              JSR HLINE
0816: CA                36              DEX
0817: 10 EF             37              BPL LOOP
                        38  ;
                        39  ; Den Bildschirm mit vert. Linien füllen
                        40  ;
0819: A0 27             41              LDY #$27        ;X-Wert
081B: 98                42  LOOP1       TYA
081C: 29 0F             43              AND #$0F        ;ausblenden
081E: 20 64 F8          44              JSR SETCOL
0821: A9 25             45              LDA #$25        ;Y-Unten
0823: 85 2D             46              STA V2
0825: A9 10             47              LDA #$10        ;Y-Oben
0827: 20 28 F8          48              JSR VLINE
082A: 88                49              DEY
082B: 10 EE             50              BPL LOOP1
                        51  ;
                        52  ; Eine „Zeile" vom Benutzer eingeben lassen
                        53  ;
```

```
082D: A2 00      54              LDX #$00
082F: AD 00 C0   55   LOOP2      LDA KBD
0832: 10 FB      56              BPL LOOP2
0834: 8D 10 C0   57              STA STROBE
0837: 9D D0 05   58              STA $05D0,X
083A: E8         59              INX
083B: E0 28      60              CPX #$28
083D: 90 F0      61              BLT LOOP2
                 62   ;
                 63   ; Schirm löschen, Bild aufbauen
                 64   ;
083F: 20 36 F8   65              JSR CLRTOP
0842: A9 0F      66              LDA #$0F        ;Weiß
0844: 20 64 F8   67              JSR SETCOL
0847: A2 00      68              LDX #$00        ;init.
0849: BD 5A 08   69   LOOP3      LDA TAB,X
084C: F0 0B      70              BEQ ENDE
084E: A8         71              TAY             ;X-Wert
084F: E8         72              INX
0850: BD 5A 08   73              LDA TAB,X       ;Y-Wert
0853: 20 00 F8   74              JSR PLOT
0856: E8         75              INX
0857: D0 F0      76              BNE LOOP3
                 77   ;
0859: 60         78   ENDE       RTS
                 79   ;
                 80   ; Tabelle X,Y.... (max. 254 Werte)
                 81   ;
085A: 02 06 02   82   TAB        HEX 0206020702080209
0862: 02 0A 02   83              HEX 020A020B020C030C
086A: 04 0C 05   84              HEX 040C050C    ;L
086E: 07 07 07   85              HEX 0707070807090709070A
0876: 07 0B 08   86              HEX 070B080C090C0A0B
087E: 0A 0A 0A   87              HEX 0A0A0A090A080A07
0886: 09 06 08   88              HEX 09060806    ;O
088A: 0C 06 0C   89              HEX 0C060C070C080C09
0892: 0C 0A 0C   90              HEX 0C0A0C0B0C0C0D06
089A: 0E 06 0F   91              HEX 0E060F070F080E09
08A2: 0D 09 0D   92              HEX 0D090D0A0E0B0F0C ;R
08AA: 14 06 13   93              HEX 1406130612061106
08B2: 11 07 11   94              HEX 1107110811091104
08BA: 11 0B 11   95              HEX 110B110C12091309
08C2: 12 0C 13   96              HEX 120C130C140C ;E
08C8: 19 06 18   97              HEX 1906180617061607
08D0: 16 08 17   98              HEX 1608170918091A0A
08D8: 19 0B 18   99              HEX 190B180C170C160C ;S
08E0: 1C 09 1D  100              HEX 1C091D091E09 ;-
08E6: 16 11 16  101              HEX 1611161216131614
08EE: 16 15 16  102              HEX 1615161616171717
08F6: 18 17 19  103              HEX 1817191619151814
08FE: 17 14 19  104              HEX 1714191319121811
0906: 17 11     105              HEX 1711        ;B
0908: 1B 11 1B  106              HEX 1B111B121B131B14
0910: 1B 15 1B  107              HEX 1B151B161B17 ;I
```

```
0916: 1D 11 1D  108           HEX 1D111D121D131D14
091E: 1D 15 1D  109           HEX 1D151D161D171E17
0926: 1F 17 20  110           HEX 1F172017    ;L
092A: 24 11 23  111           HEX 2411231122112212
0932: 22 13 22  112           HEX 2213221422152216
093A: 22 17 23  113           HEX 2217231724172516
0942: 25 15 25  114           HEX 2515251425132512 ;D
094A: 00        115           HEX 00          ;ENDE
```

Zwei weitere LORES-Routinen haben wir nicht benötigt:

NEXTCOL $F85F
Erhöht die COLOR-Nummer um 3.

Eingabe: –

Ausgabe: COLOR-Maske in beiden Nibbles um 3 erhöht

SCRN $F871
Bestimmt die Farbe eines LORES-Punktes X,Y.

Eingabe: Akku = Y, Y-Reg = X

Ausgabe: Akku = Farbnummer des Punktes

3. Der kleinkarierte Apfel

Der Monitor enthält keine Routinen für die hochauflösende Grafik des Apple, so daß wir für unsere nächsten Versuche fast ausschließlich auf den Applesoft-Interpreter angewiesen sind. Ein Listing ist von Apple nie veröffentlicht worden, aber freie Autoren haben dies nachgeholt, indem sie den Maschinencode von $D000 bis $F7FF analysierten. In bezug auf die HIRES-Routinen sind alle Applesoft-Interpreter – vom Apple IIplus bis zum IIc – identisch, so daß die hier aufgelisteten Informationen als „sicher" gelten können.
Wir wollen in diesem Buch nicht über die theoretischen und programmierpraktischen Prinzipien der Interpreter-Routinen reden, da allein die Theorie über das Ziehen einer Diagonalen auf einer Rasterseite ein paar Buchseiten füllt. Stattdessen werden wir lernen, die Routinen mit den richtigen Parametern aufzurufen und so alle Applesoft-Befehle auch von der Assembler-Ebene auszuführen.

3.1 Hilfe, wo steht der Punkt

Die hochauflösende Grafik benutzt einen anderen Speicherbereich als die Text- oder LORES-Anzeige. HIRES 1 (HGR1) belegt $2000 bis $3FFF und HIRES 2 (HGR2) benötigt die Speicherplätze von $4000 bis $5FFF für sich. Die hochauflösende Grafik nutzt also zwei RAM-Blöcke von je 8 KByte, die heute mitten im Hauptspeicher liegen. Vor einigen Jahren hatte der Apple nur 16K RAM, und da lag die Grafik ganz oben. Der Speicher ist heute auf 128K gewachsen, aber da die ROM-Routinen noch immer die selben geblieben sind, müssen wir jetzt unsere Programme um die HIRES-Grafik herumschreiben.
Wenn Sie die Organisation des Textbildschirms schon kompliziert fanden, dann werden Sie beim HIRES-Schirm vielleicht ein verzweifeltes Lächeln aufsetzen. Bei näherem Hinsehen sind aber viele Parallelen zum Textbildschirm zu finden, so daß wir auch die HIRES „in den Griff bekommen".

Der Apple hat eine Auflösung von 280 * 192 Punkten. Mit der doppelthochauflösenden Grafik des IIc und des IIe mit 64k-Karte sind daneben noch 560 * 192 Punkte möglich. Wir werden an dieser Stelle aber auf die Double-HIRES nicht weiter eingehen. In der Zeitschrift „Peeker" sind einige Aufsätze über Double-HIRES erschienen. Um zu DHGR zu kommen, müssen Sie die folgenden Softswitches betätigen:

```
STA $C050 TXTCLR
STA $C052 MIXCLR
STA $C057 HIRES
STA $C001 80STORE
STA $C00D 80COL
STA $C05E DHIRESON      STA C05F DHIRESOFF
```

Wenn Sie die 192 Punkte, die von 0 bis 191 (= $00 – $BF) gezählt werden, in Gruppen zu je 8 Zeilen aufteilen, erhalten Sie 24 Gruppen. Diese entsprechen den 24 Zeilen des Textbildschirms. Wenn Sie die jeweils oberste Linie als „Grundlinie" bezeichnen, so folgen die 7 nächsten Linien jeweils mit einer um $0400 höheren Adresse im Speicher (siehe Abb. 4). Während beim LORES aus einer Textzeile zwei Grafikzeilen werden, die durch die beiden Nibble jedes Bytes adressiert werden, spaltet sich bei der HIRES-Grafik jede Gruppe in 8 Grafikzeilen auf, die alle ihre eigenen Speicherplätze besitzen.

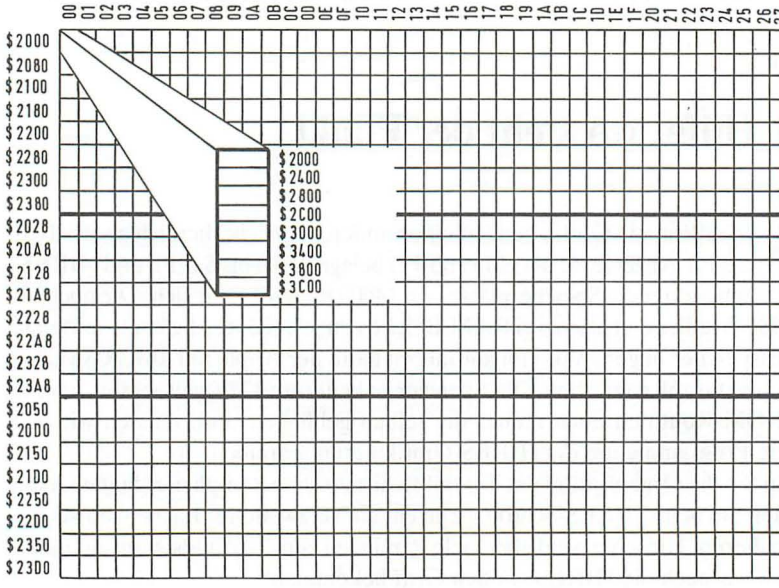


Abb. 4: Adressierung der hochauflösenden Grafikseite 1

Wenn Sie sich jetzt die Gruppen als Textzeilen, die die Adresse der Grundlinie der Gruppe tragen, vorstellen, so haben Sie den Rest der Organisation schon begriffen. Die Grundlinien lassen sich wiederum in 3 Blöcke unterteilen. Zu Block 1 gehören die Zeilen $00 bis $3F, zu Block 2 die Zeilen $40 bis $7F und zu Block 3 die Zeilen $80 bis $BF. Diese sind den drei Blöcken der Textseite vergleichbar (siehe Abb. 1). Die Zeilen $00, $40 und $80 sind die Grundlinien der Blöcke. Ihre Startadressen sind $2000, $2028 und $2050 ($4000, $4028 und $4050 für HGR2). Der Abstand von $28 Bytes entspricht genau dem Abstand der Textzeilen 0, 8 und 16, die die Startlinien der Textblöcke sind.
Die Grundlinien der Gruppen innerhalb eines Blocks sind regelmäßig angeordnet und gegeneinander um $80 Bytes versetzt, genau wie die Textzeilen innerhalb eines Blocks.
Die Analogie zum Textbildschirm geht so weit, daß auch beim HIRES-Schirm Scratch-Bytes existieren. Nur sind es hier achtmal so viele, insgesamt also 512 Bytes, die nicht angezeigt werden. Sie liegen jeweils „am Ende" der Zeilen im dritten (unteren) Block. Im normalen Grafik-Betrieb werden sie nicht benutzt, stehen uns also als Zwischenspeicher zur Verfügung. Manche Autoren benutzen sie, um zeitweise nicht sichtbare Grafik-Objekte dort zu „parken". Diese Speicherung ist nicht dauerhaft, da jeder HGR-Befehl bzw. sein Assembler-Gegenstück auch diese unsichtbaren Bytes mit löscht. Die Scratch-Bytes sind auch dafür verantwortlich, daß Sie ein HGR-Bild nur bis $3FF8 bzw. $5FF8 abzuspeichern brauchen. Ohne Informationsverlust benötigt das Binärfile unter DOS 3.3 dann nur noch 33 und nicht mehr 34 Sektoren.

Einen Aspekt haben wir bisher nicht berücksichtigt: wie liegen die Punkte einer Zeile im Speicher? Zunächst einmal die gute Nachricht: zu jeder Zeile gehören 40 Bytes, die hintereinander im Speicher angeordnet sind. Das entspricht dem Textbildschirm. Zu jedem Punkt auf dem Bildschirm gehört 1 Bit im Speicher. Ein Punkt ist auf dem Bildschirm sichtbar (an), wenn das zugehörige Bit gesetzt (1) ist.
Und jetzt die zwei schlechten Nachrichten:
1. Von jedem Byte werden nur die unteren 7 Bits angezeigt. Das Bit 7 (das 8. Bit!) wird benutzt, um die Farbe der übrigen 7 Bits festzulegen. So kommt auch die horizontale Auflösung von $7 * 40 = 280$ Punkte zustande.
2. Die Bitnummern im Speicher und auf dem Bildschirm sind genau umgekehrt angeordnet. Während Bit 0 im Speicher ganz rechts steht, findet sich der zugehörige Punkt ganz links auf dem Bildschirm.

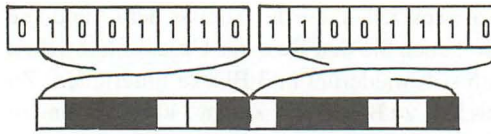


Abb. 5: Zuordnung von Bitmuster und Bildschirmpunkten

3.2 Was wären wir ohne ROM!

Werfen Sie nicht gleich das Handtuch, wenn Sie die HIRES-Organisation nicht sofort verstanden haben. Alle notwendigen Funktionen, um jeden HIRES-Punkt setzen oder löschen zu können, sind fertig im ROM vorhanden. Zu diesen Routinen gehört ein Satz von Zero-Page-Variablen, die viele Parameter festlegen:

Byte	Funktion
$1A SHAPEL	Lo-Byte des Shapeanfangs
$1B SHAPEH	Hi-Byte des Shapeanfangs
$1C HCOLOR1	aktive Farbmaske
$26 GBASL	Lo-Grundadresse einer Grafikzeile
$27 GBASH	Hi-Grundadresse einer Grafikzeile
$30 HMASK	Bitmaske für einen Punkt
$E0 X0L	Lo-Byte der horizontalen Koordinate (XLo)
$E1 X0H	Hi-Byte der horizontalen Koordinate (XHi)
$E2 Y0	vertikale Koordinate (Y)
$E4 HCOLORZ	Farbmaske des HCOLOR-Befehls
$E5 HNDX	horizontaler Offset (= welches Byte der Zeile)
$E6 HPAG	Seitenanzeige ($20 = HGR1, $40 = HGR2)
$E7 SCALEZ	Skalierungsfaktor für Shapes
$E8 SHAPEPNT	Lo-Byte Zeiger auf Shape-Table-Anfang
$E9 ---	Hi-Byte Zeiger auf Shape-Table-Anfang
$EA COLCOUNT	Kollisionszähler

Um die HGR-Seiten anzuzeigen, müssen Sie ein paar Softswitches betätigen. Die Reihenfolge ist wiederum unerheblich.

```
HIRES Seite 1 (HGR1)        HIRES Seite 2 (HGR2)

   LDA TXTCLR                   LDA TXTCLR
   LDA MIXCLR                   LDA MIXCLR
   LDA LOWSCR                   LDA HISCR
   LDA HIRES                    LDA HIRES
```

In beiden Fällen wird der volle Grafik-Schirm eingeschaltet. Durch MIXSET anstelle von MIXCLR können Sie wie bei der LORES-Grafik unten 4 Textzeilen einblenden. Bei HGR1 gehören diese zu TEXT1, bei HGR2 zu TEXT2! Dieser „gepokete" Aufruf löscht die vorhandene Grafik nicht.

Wenn Sie die Grafikseiten **mit** Löschen aufrufen wollen, benutzen Sie die Applesoft-Funktionen HGR ($F3E2) bzw. HGR2 ($F3D8). HGR setzt die Softswitches für HGR1 und gemischten Schirm. Nach HPAG wird $20 geschrieben und die Farbe HCOLORZ wird auf $00 (Schwarz) gesetzt. Der Bereich von $2000 bis $3FFF wird auf Null gesetzt. HGR2 führt dasselbe für die zweite Seite aus, nur daß diesmal auf den vollen Grafikschirm geschaltet wird. Wollen Sie auch die Seite 1 ganz sehen, so müssen Sie

```
JSR HGR
STA MIXCLR
```

ausführen.

HGR $F3E2
Wählt HGR1, gemischter Schirm. Löscht den Schirm.

Eingabe: –

Ausgabe: HPAG = $20, HCOLOR1 = $00

HGR2 $F3D8
Wählt HGR2, voller Schirm. Löscht den Schirm.

Eingabe: –

Ausgabe: HPAG = $40, HCOLOR1 = $00

Sie können die Löschroutine auch separat aufrufen über HCLR ($F3F2). Durch einen späteren Einstieg bei BKGND ($F3F6) ist es möglich, den Bildschirm in der gewünschten Farbe einzufärben.

HCLR $F3F2
Löscht den aktiven HIRES-Schirm.

Eingabe: HPAG = $20 für HGR1, $40 für HGR2

Ausgabe: HCOLOR = $00

BKGND $F3F6
Färbt aktiven HIRES-Schirm ein.

Eingabe:
HPAG = $20 für HGR1, $40 für HGR2
HCOLOR1 = gewünschter Farbcode

Ausgabe: –

3.3 Eine Linie verschwindet

Zur Farbenpracht der HIRES-Grafik habe ich mich bisher ausgeschwiegen. Das hat seinen guten Grund. Der Apple kennt 8 Farben. Vier davon sind paarweise gleich: Weiß1 und Weiß2 sowie Schwarz1 und Schwarz2. Somit bleiben 6 unterscheidbare Farben. Nicht jeder Punkt kann aber alle 6 Farben annehmen. Wie so häufig bei der Apple-Grafik liegen die Verhältnisse komplizierter. Jedes Bit kann in einer von zwei Farben erscheinen. Diese Farben sind Violett oder Blau, wenn der Punkt eine gerade Spaltennummer besitzt. Bei einer ungeraden Spaltennummer besteht die Auswahl aus Grün und Orange. Die Farben Violett und Grün erscheinen, wenn das Farbbit (= Bit 7) des Bytes nicht gesetzt (0) ist. Bei gesetztem Farbbit (1) sehen wir Blau und Orange. Eine Zeile besteht also

immer abwechselnd aus violetten und grünen sowie aus blauen und orangen Punkten.
Nehmen wir zunächst an, das Farbbit sei gelöscht. Wenn Sie den Punkt in Spalte 0 anschalten, erscheint er Violett. Ein Punkt in Spalte 1 wird Grün wiedergegeben. Wenn Sie gleichzeitig sowohl den Punkt in Spalte 0 als auch den in Spalte 1 anschalten, verschmelzen diese zu einer kurzen horizontalen Linie, die Weiß erscheint. Allgemein gilt, daß zwei benachbarte Punkte zusammen weiß ergeben. Wenn Sie eine violette Linie zeichnen wollen, dürfen Sie also nur die Punkte in den geraden Spalten einschalten. Das ist leichter gesagt als getan. Da 7 Punkte zu einem Byte gehören, wechselt die Anfangsfarbe von Byte zu Byte. Die geradzahligen Bytes beginnen mit Violett, die ungeradzahligen dagegen mit Grün. Eine grüne Linie beginnt mit folgendem Punktemuster:

```
  V G V G V G V       G V G V G V G       V G V G V G V
    *   *   *         *   *   *   *         *   *   *
```

daraus ergeben sich die folgenden Speicherbytes:

```
0 0 1 0 1 0 1 0     0 1 0 1 0 1 0 1     0 0 1 0 1 0 1 0
      $2A                 $55                 $2A
```

Wie Sie sehen, sind die Bytes nicht identisch. Das mittlere Byte muß einmal nach rechts geshiftet (LSR) werden, um den Wert seiner Nachbarn zu erhalten. Wenn Sie das Farbbit unberücksichtigt lassen, können Sie die beiden Formen durch einfaches Invertieren ineinander überführen (EOR $7F).
Wenn wir die Zeichenroutinen des Applesoft benutzen, wird die Verschiebung vom Programm berechnet. Dies ist auch der Grund für die vielen Bit- und Bytemasken auf der Zero-Page.

Da es für jeweils 7 Punkte nur ein Farbbit gibt, ist die Farbauswahl noch weiter eingeschränkt. So kann es z.B. keine grüne und orange Linie im gleichen Byte geben. Dies Problem betrifft besonders vertikale Linien. Versuchen Sie einmal folgendes kurze BASIC-Programm:

FARBTEST

```
10  REM  Farbtest
20  HGR2 : HCOLOR= 1: REM  Grün
30  HPLOT 1,0 TO 1,191
40  GET A$: REM  Tastendruck
50  HCOLOR= 5: REM  Orange
60  HPLOT 5,0 TO 5,191
70  GET A$: REM  Tastendruck
80  HPLOT 8,0 TO 8,191
90  GET A$: TEXT : END
```

Zunächst wird eine vertikale grüne Linie gezeichnet. Dann folgt nach einem Tastendruck daneben eine orange Linie. Wenn Sie einen Farbmonitor besitzen,

können Sie erkennen, daß dabei die grüne Linie auch orange wird. Besitzer von Schwarz/Weiß-Monitoren können bei genauem Hinsehen erkennen, daß die erste Linie ein klein wenig nach rechts rutscht. Nach einem erneuten Tastendruck sollte eine weitere vertikale Linie erscheinen. Sie werden aber vergeblich auf sie warten: die Linie bleibt verschwunden.
Für die erste grüne Linie mußte das jeweilige Farbbit gelöscht sein. Die orange Linie benötigt dagegen ein gesetztes Farbbit. Da die Spalten 1 und 5 zum selben Byte gehören, wurde damit auch für die erste Linie das Farbbit gesetzt: sie wurde orange. Die dritte Linie haben wir in einer geradzahligen Spalte gezeichnet. Dort sind aber nur die Farben Blau und Violett sichtbar. Unsere orange Linie bleibt im Verborgenen.

Die gewünschte Farbe eines Punktes oder einer Linie wird durch die Speicherstelle HCOLORZ ($00E4) bestimmt. Um dort das richtige Bitmuster zu setzen, benutzen wir die Routine SETHCOL ($F6EC). Die Farbnummer wird im X-Register übergeben. Liegt der Wert über 7, wird ein „Illegal Quantity Error" ausgegeben. Wenn Sie möchten, können Sie HCOLORZ auch direkt mit einem Farbcode setzen.

HCOLOR Index

HCOLOR	Farbcode		Farbe
0	$00	00000000	Schwarz 1
1	$2A	00101010	Grün
2	$55	01010101	Violett
3	$7F	01111111	Weiß 1
4	$80	10000000	Schwarz 2
5	$AA	10101010	Orange
6	$D5	11010101	Blau
7	$FF	11111111	Weiß 2

SETHCOL $F6EC
Setzt Colormaske für Farben 1 bis 7

Eingabe: X-Reg = HCOLOR (1..7)

Ausgabe: HCOLORZ = Farbcode

3.4 Wiederaufführung

Für das Zeichnen von Punkten und einfachen Linienfolgen wollen wir uns noch einmal das Programm „BEISPIEL NR. 3B“ auf Seite 18 des ersten Bandes ansehen. Dort wird zunächst die zweite HIRES-Seite aufgerufen und gelöscht. Anschließend wird die Farbe Blau direkt gesetzt.
Zum Zeichnen von Punkten und Linien gibt es in Applesoft nur einen Befehl: HPLOT. In Assembler müssen wir dagegen zwei Routinen benutzen: HPLOT ($F457) und HGLIN ($F53A). HPLOT setzt einen einzigen Punkt. HGLIN zieht eine Verbindungsline vom letzten gesetzten Punkt zu einem angegebenen Endpunkt. Das bedeutet, daß vor dem Aufruf von HGLIN wenigstens einmal auch HPLOT aufgerufen werden mußte, damit ein Anfangspunkt existiert. Verschiedene HGLIN-Aufrufe können anschließend aneinandergereiht werden. Unser Beispielprogramm zeichnet ein Rechteck, indem zunächst ein Eckpunkt mit HPLOT festgelegt wird. Von ihm aus werden dann die vier Seiten gezogen.

HPLOT $F457
Setzt einen Punkt X,Y

Eingabe:
HPAG = $20 oder $40, HCOLORZ gesetzt
Akku = Y, X-Reg = XLo, Y-Reg = XHi

Ausgabe:
HCOLOR1 = Farbmaske
GBASL/H zeigt auf Startadresse der zu X,Y gehörenden Zeile
X0L/H, Y0 Koordinate von X,Y
HMASK = Bitmaske, Y-Reg = HNDX = Spaltenindex

HGLIN $F53A
Zieht eine Verbindungslinie zwischen zwei Punkten

Eingabe:
X0L/H = X-Koordinate des 1. Punktes
Y0 = Y-Koordinate des 1. Punktes
HPAG = $20 oder $40
HCOLORZ richtig gesetzt
Akku = XLo des 2. Punktes, X-Reg = XHi des 2. Punktes
Y-Reg = Y des 2. Punktes

Ausgabe:
HPAG und HCOLORZ unverändert, HCOLOR1 gesetzt
X0L/H und Y0 = Koordinaten des 2. Punktes
GBASL/H = Startadresse der zuletzt bearbeiteten Zeile
HNDX = Spaltenindex des Endpunktes

Wenn Sie die Zero-Page-Variablen setzen wollen, ohne einen sichtbaren Punkt zu zeichnen, können Sie das mit HPOS ($F411) tun und damit den (unsichtbaren) HIRES-Cursor positionieren. Mit HFIND ($F5CB) können Sie die internen Cursor-Daten wieder in X- und Y-Werte zurückrechnen.

HPOS $F411
Berechnet die internen HIRES-Cursor-Werte

Eingabe:
HPAG = $20 oder $40, HCOLORZ gesetzt
Akku = Y, X-Reg = XLo, Y-Reg = XHi

Ausgabe:
Akku = HCOLOR1 = Farbmaske
GBASL/H zeigt auf Startadresse der zu X,Y gehörenden Zeile
X0L/H, Y0 Koordinaten von X,Y
HMASK = Bitmaske, Y-Reg = HNDX = Spaltenindex

HFIND $F5CB
Berechnet X,Y aus den internen Cursordaten

Eingabe:
HBASL/H = Zeilenadresse
HNDX = Spaltenindex
HMASK = Bitmaske

Ausgabe:
X0L/X0H = X-Koordinate
Y0 = Y-Koordinate

3.5 Eine Tabelle mit Bildern

Einen letzten Punkt haben wir noch nicht besprochen: die Benutzung von Shapes (Formtabellen). Von Basic aus können Sie durch „DRAW I AT X,Y“ ganze Linienfolgen auf den HIRES-Schirm bringen. Dazu müssen Sie vorher eine Shapetabelle anlegen, deren Aufbau im Applesoft-Handbuch beschrieben ist. Sie können diese Technik auch in Assembler nutzen. Der BASIC-Befehl „XDRAW“ ist ebenfalls vorhanden.

Sowohl DRAW ($F601) als auch XDRAW ($F65D) werden in identischer Weise aufgerufen. Zunächst muß mit HPOS der interne Cursor auf den Startpunkt der Shape-Zeichnung gesetzt werden. Die Speicherstelle SCALEZ ($00E7) muß gesetzt und der Akkumulator mit dem Rotationswert geladen werden. Im X- und Y-Register sind die Startadresse der Shapedefinition (nicht der Shapetabelle!!) zu übergeben.

DRAW $F601
Zeichnet ein Shape durch OR-Verknüpfung mit dem Hintergrund.

Eingabe:
Interne Cursordaten gesetzt (durch HPOS)
X-Reg = Lo-Byte des Shapeanfangs
Y-Reg = Hi-Byte des Shapeanfangs
Akku = Rotationswert, SCALEZ = Skalierungsfaktor

Ausgabe:
COLCOUNT = $00, wenn keine Kollision mit gesetzten Bits
sonst COLCOUNT = Zahl der Kollisionen

XDRAW $F65D
Zeichnet ein Shape durch XOR (= EOR)-Verknüpfung mit dem Hintergrund.

Eingabe:
Interne Cursordaten gesetzt (durch HPOS)
X-Reg = Lo-Byte des Shapeanfangs
Y-Reg = Hi-Byte des Shapeanfangs
Akku = Rotationswert, SCALEZ = Skalierungsfaktor

Ausgabe:
COLCOUNT = $00, wenn keine Kollision mit gesetzten Bits
sonst COLCOUNT = Zahl der Kollisionen

Die Routinen DRAW und XDRAW haben den Nachteil, daß Sie immer die Anfangsadresse des gewünschten Shapes wissen müssen. Bei langen Shape-Ta-

bellen müssen Sie dann einen langen Merkzettel auf Ihrem Tisch liegen haben. Viel wahrscheinlicher ist, daß Sie den Beginn der Shape-Tabelle wissen und die Nummer des Shapes in der Tabelle. Im Applesoft-ROM existiert zwar eine Routine, die die Adresse eines Shapes berechnet, wir können sie aber nicht nutzen, da sie versucht, aus einem BASIC-Programm weitere Informationen zu lesen. Einen Teil dieser Routine bilden wir deshalb in unseren eigenen Programmen als Unterroutine nach.

DRW PARA

```
                        1   ;**************************
                        2   ;        DRWPARA          *
                        3   ;  Berechnet die Anfangs- *
                        4   ; adresse eines Shapes in *
                        5   ;  einer bekannten Shape- *
                        6   ;tabelle. Dem ROM entlehnt*
                        7   ;                         *
                        8   ; Aufruf: SHAPEPNT auf Be-*
                        9   ; ginn der Tabelle gesetzt*
                        10  ;    X-Reg = Shapenummer  *
                        11  ;**************************
                        12  ;
                        13  SHAPEL    EQU $001A
                        14  SHAPEH    EQU $001B
                        15  SHAPEPNT  EQU $00E8      ;+$00E9
                        16  ;
                        17  ILQERR    EQU $E199
                        18  ;
                        19            ORG $300       ;nur Beispiel!
                        20  ;
0300: A5 E8             21  DRWPARA   LDA SHAPEPNT
0302: 85 1A             22            STA SHAPEL
0304: A5 E9             23            LDA SHAPEPNT+1
0306: 85 1B             24            STA SHAPEH
0308: 8A                25            TXA
0309: A2 00             26            LDX #$00
030B: D2 1A             27            CMP (SHAPEL),X ;Maximalzahl
030D: F0 05             28            BEQ NUMOK      ;=
030F: 90 03             29            BCC NUMOK      ;<
0311: 4C 99 E1          30            JMP ILQERR     ;gibt es nicht
                        31  ;
0314: 0A                32  NUMOK     ASL            ;*2
0315: 90 03             33            BCC KLEIN      ;Shape < 128
0317: E6 1B             34            INC SHAPEH
0319: 18                35            CLC
031A: A8                36  KLEIN     TAY            ;auf Abstandstab.
031B: B1 1A             37            LDA (SHAPEL),Y ;Abstand vom Ta-
031D: 65 1A             38            ADC SHAPEL     ;bellenanf. add.
031F: AA                39            TAX            ;retten
0320: C8                40            INY
0321: B1 1A             41            LDA (SHAPEL),Y
0323: 65 E9             42            ADC SHAPEPNT+1 ;urspr. Wert
0325: 85 1B             43            STA SHAPEH
0327: 86 1A             44            STX SHAPEL
0329: 60                45            RTS
```

Da nach dieser Routine SHAPEL/H schon korrekt gesetzt ist, springen Sie in DRAW bzw. XDRAW etwas später ein. Das Laden von X-und Y-Register entfällt. **DRAW1** hat die Adresse $F605, **XDRAW1** ist unter $F661 zu erreichen.

3.6 Punkte werden mobil

Für viele Anwendungen – besonders für Spiele – müssen Sie den HIRES-Schirm schnell in alle 4 Richtungen verschieben. Da dabei immer 8KByte bewegt werden müssen, ist Assembler die einzige Sprache, die dieser Aufgabe auf dem Apple gewachsen ist. Das folgende Programm ist eine sehr schnelle Routine, um den gesamten HIRES-Bildschirm (HGR1) um einen Punkt nach rechts, links, oben und unten zu verschieben. Außerdem gibt es die Möglichkeit, horizontal um 1 Byte (7 Punkte) schnell zu verschieben. Für viele Spiele ist das ausreichend, da das auftretende „Rucken" in der schnellen Bewegung meistens untergeht. Die Geschwindigkeit ist nur zu erreichen, indem die Anfangsadressen der einzelnen Grafikzeilen einer Tabelle entnommen werden. Eine Berechnung wäre zu zeitaufwendig. Damit Sie dieses Programm auch von BASIC aus nutzen können, liegen am Anfang feste Sprungadressen zu den einzelnen Unterroutinen, die Sie mit einem CALL ... anspringen.
(TIEF = 36864, HOCH = TIEF+3, RECHTS = HOCH+3, LNKS = RECHTS+3, FR = LNKS+3, FL = FR+3 ... ... CALL TIEF CALL FR usw.)
In BASIC müssen Sie zum Teil andere Namen wählen, damit Applesoft darin keine reservierten Worte entdeckt.

HIRES-SCROLL

```
1    ;*********************************
2    ;       * HIRES-SCROLL *         *
3    ;     * alle 4 Richtungen *      *
4    ;     * um je einen Punkt *      *
5    ;     *  FAST je 1 Byte   *      *
6    ;     *  rechts o. links  *      *
7    ;     *********************       *
8    ; (C) 1985  Dr. Jürgen B. Kehrel *
9    ;*********************************
10   ;
11   ;
12             ORG $9000
13   ;
14   PTR       EQU $0000
15   PTR1      EQU $0002
16   ;
17   ; Vektoren zu den einzelnen Routinen
18   ;
```

```
9000: 4C 12 90   19  START     JMP TIEF
9003: 4C 41 90   20            JMP HOCH
9006: 4C 70 90   21            JMP RECHTS
9009: 4C 9F 90   22            JMP LINKS
900C: 4C CB 90   23            JMP FASTR
900F: 4C EF 90   24            JMP FASTL
                 25  ;
                 26  ; Verschiebung nach unten
                 27  ; oberste Zeile ($2000ff.) löschen
                 28  ;
9012: A2 00      29  TIEF      LDX #$00        ;unterste Zeile
9014: BD 15 91   30  TL1       LDA TABL0,X     ;Adresse Lo-Byte
9017: 85 00      31            STA PTR         ;merken
9019: BD D5 91   32            LDA TABHI,X     ;Adresse Hi-Byte
901C: 85 01      33            STA PTR+1       ;merken
901E: E8         34            INX             ;Zeile darüber
901F: BD 15 91   35            LDA TABL0,X     ;Adresse Lo-Byte
9022: 85 02      36            STA PTR1        ;merken
9024: BD D5 91   37            LDA TABHI,X     ;Adresse Hi-Byte
9027: 85 03      38            STA PTR1+1      ;merken
9029: A0 27      39            LDY #$27        ;$28 (40) Bytes
902B: B1 02      40  TL2       LDA (PTR1),Y    ;Byte laden
902D: 91 00      41            STA (PTR),Y     ;1 Z. tiefer sp.
902F: 88         42            DEY             ;nächstes Byte
9030: 10 F9      43            BPL TL2         ;Zeile fertig?
9032: E0 BF      44            CPX #$BF        ;alle Z. fertig?
9034: D0 DE      45            BNE TL1         ;nein, neue Adr.
9036: A9 00      46            LDA #$00        ;oberste Z. löschen
9038: A0 27      47            LDY #$27        ;auch $28 Bytes (40)
903A: 99 00 20   48  TL3       STA $2000,Y     ;Schleife
903D: 88         49            DEY             ;nächstes Byte
903E: 10 FA      50            BPL TL3         ;sind wir fertig?
9040: 60         51            RTS             ;Ja, zurück
                 52  ;
                 53  ; Verschiebung nach oben
                 54  ; unterste Zeile ($3FD0ff.) löschen
                 55  ;
9041: A2 BF      56  HOCH      LDX #$BF        ;oberste Zeile
9043: BD 15 91   57  HL1       LDA TABL0,X     ;Adresse Lo-Byte
9046: 85 00      58            STA PTR         ;merken
9048: BD D5 91   59            LDA TABHI,X     ;Adresse Hi-Byte
904B: 85 01      60            STA PTR+1       ;merken
904D: CA         61            DEX             ;Zeile darunter
904E: BD 15 91   62            LDA TABL0,X     ;Adresse Lo-Byte
9051: 85 02      63            STA PTR1        ;merken
9053: BD D5 91   64            LDA TABHI,X     ;Adresse Hi-Byte
9056: 85 03      65            STA PTR1+1      ;merken
9058: A0 27      66            LDY #$27        ;$28 (40) Bytes
905A: B1 02      67  HL2       LDA (PTR1),Y    ;Byte laden und
905C: 91 00      68            STA (PTR),Y     ;1 Z. höher sp.
905E: 88         69            DEY             ;nächstes Byte
905F: 10 F9      70            BPL HL2         ;Zeile fertig?
9061: E0 00      71            CPX #$00        ;alle Z. fertig?
9063: D0 DE      72            BNE HL1         ;Nein, nächste Z.
```

```
9065: A9 00      73              LDA #$00        ;unterste Z. löschen
9067: A0 27      74              LDY #$27        ;$28 (40) Bytes
9069: 99 D0 3F   75   HL3        STA $3FD0,Y     ;Schleife
906C: 88         76              DEY             ;nächstes Byte
906D: 10 FA      77              BPL HL3         ;fertig?
906F: 60         78              RTS             ;Ja, zurück
                 79   ;
                 80   ; Verschiebung nach rechts
                 81   ; Farbbit bleibt konstant
                 82   ;
9070: A2 00      83   RECHTS     LDX #$00        ;unterste Zeile
9072: BD 15 91   84   RL1        LDA TABL0,X     ;Adresse Lo-Byte
9075: 85 00      85              STA PTR         ;merken
9077: BD D5 91   86              LDA TABHI,X     ;Adresse Hi-Byte
907A: 85 01      87              STA PTR+1       ;merken
907C: A0 00      88              LDY #$00        ;links beginnen
907E: 84 02      89              STY PTR1        ;Übertrag löschen
9080: B1 00      90   RL2        LDA (PTR),Y     ;Byte laden
9082: 0A         91              ASL             ;Farbe -> Carry
9083: 24 02      92              BIT PTR1        ;Übertrag?
9085: 10 02      93              BPL RL3         ;Nein
9087: 09 01      94              ORA #$01        ;Ja, einsetzen
9089: 85 02      95   RL3        STA PTR1        ;Übertrag merken
908B: B0 03      96              BCS RL4         ;war Farbbit gesetzt?
908D: 29 7F      97              AND #$7F        ;Nein, Bit7 löschen
908F: 2C         98              HEX 2C          ;BIT, Trick
9090: 09 80      99   RL4        ORA #$80        ;Ja, Bit7 setzen
9092: 91 00      100             STA (PTR),Y     ;Byte zurücksp.
9094: C8         101             INY             ;nächstes Byte
9095: C0 28      102             CPY #$28        ;Zeile fertig?
9097: D0 E7      103             BNE RL2         ;Nein, weiter
9099: E8         104             INX             ;nächste Zeile
909A: E0 C0      105             CPX #$C0        ;alle Z. fertig?
909C: D0 D4      106             BNE RL1         ;Nein, neue Adresse
909E: 60         107             RTS             ;Ja, zurück
                 108  ;
                 109  ; Verschiebung nach links
                 110  ; Farbbit bleibt konstant
                 111  ;
909F: A2 00      112  LINKS      LDX #$00        ;unterste Zeile
90A1: BD 15 91   113  LL1        LDA TABL0,X     ;Adresse Lo-Byte
90A4: 85 00      114             STA PTR         ;merken
90A6: BD D5 91   115             LDA TABHI,X     ;Adresse Hi-Byte
90A9: 85 01      116             STA PTR+1       ;merken
90AB: A0 27      117             LDY #$27        ;rechts beginnen
90AD: 18         118             CLC             ;Übertrag löschen
90AE: B1 00      119  LL2        LDA (PTR),Y     ;Byte laden
90B0: 85 02      120             STA PTR1        ;merken f. Farbbit
90B2: 90 03      121             BCC LL3         ;Übertrag vorhanden?
90B4: 09 80      122             ORA #$80        ;Ja, einsetzen
90B6: 2C         123             HEX 2C          ;BIT, Trick
90B7: 29 7F      124  LL3        AND #$7F        ;Nein, löschen
90B9: 4A         125             LSR             ;schieben
90BA: 24 02      126             BIT PTR1        ;war Farbbit ges.?
```

```
90BC: 10 02      127             BPL LL4         ;Nein
90BE: 09 80      128             ORA #$80        ;Ja, Bit7 setzen
90C0: 91 00      129  LL4        STA (PTR),Y     ;Byte zurückschreiben
90C2: 88         130             DEY             ;nächstes Byte
90C3: 10 E9      131             BPL LL2         ;Zeile fertig?
90C5: E8         132             INX             ;nächste Zeile
90C6: E0 C0      133             CPX #$C0        ;alle Zeilen fertig?
90C8: D0 D7      134             BNE LL1         ;Nein, weiter
90CA: 60         135             RTS             ;Ja, zurück
                 136  ;
                 137  ; Schnelle Verschiebung nach rechts
                 138  ;
90CB: A2 00      139  FASTR      LDX #$00        ;unterste Zeile
90CD: BC 15 91   140  FR1        LDY TABL0,X     ;Adresse Lo-Byte
90D0: 84 00      141             STY PTR         ;merken
90D2: BD D5 91   142             LDA TABHI,X     ;Adresse Hi-Byte
90D5: 85 01      143             STA PTR+1       ;merken
90D7: C8         144             INY             ;Adresse + 1
90D8: 84 02      145             STY PTR1        ;merken
90DA: 85 03      146             STA PTR1+1      ;Hi-Byte (k. Übertr.)
90DC: A0 26      147             LDY #$26        ;vorletzes B. beginnt
90DE: B1 00      148  FR2        LDA (PTR),Y     ;Byte laden und
90E0: 91 02      149             STA (PTR1),Y    ;1 Pos. weiter rechts
90E2: 88         150             DEY             ;nächstes Byte
90E3: 10 F9      151             BPL FR2         ;Zeile fertig?
90E5: C8         152             INY             ;auf "0" zurück
90E6: 98         153             TYA             ;Akku auch
90E7: 91 00      154             STA (PTR),Y     ;letzes Byte löschen
90E9: E8         155             INX             ;nächste Zeile
90EA: E0 C0      156             CPX #$C0        ;alle Zeilen fertig?
90EC: 90 DF      157             BLT FR1         ;Nein, weiter
90EE: 60         158             RTS             ;Ja, zurück
                 159  ;
                 160  ; Schnelle Verschiebung nach links
                 161  ;
90EF: A2 00      162  FASTL      LDX #$00        ;unterste Zeile
90F1: BC 15 91   163  FL1        LDY TABL0,X     ;Adresse Lo-Byte
90F4: 84 00      164             STY PTR         ;merken
90F6: BD D5 91   165             LDA TABHI,X     ;Adresse Hi-Byte
90F9: 85 01      166             STA PTR+1       ;merken
90FB: C8         167             INY             ;Adresse + 1
90FC: 84 02      168             STY PTR1        ;merken
90FE: 85 03      169             STA PTR1+1      ;auch Hi-Byte
9100: A0 00      170             LDY #$00        ;links beginnen
9102: B1 02      171  FL2        LDA (PTR1),Y    ;Byte laden und
9104: 91 00      172             STA (PTR),Y     ;1 Pos. nach links
9106: C8         173             INY             ;nächstes Byte
9107: C0 27      174             CPY #$27        ;Zeilenende?
9109: 90 F7      175             BLT FL2         ;Nein, weiter
910B: A9 00      176             LDA #$00        ;letzes B. löschen
910D: 91 00      177             STA (PTR),Y
910F: E8         178             INX             ;nächste Zeile
9110: E0 C0      179             CPX #$C0        ;alle Zeilen fertig?
9112: 90 DD      180             BLT FL1         ;Nein, weiter
```

```
9114: 60          181           RTS            ;Ja, zurück
                  182  ;
                  183  ; Tabelle der Zeilenadressen
                  184  ; von unten nach oben HGR 1
                  185  ;
9115: D0 D0 D0    186  TABL0     HEX D0D0D0D0D0D0D0D0 ;Lo-Bytes
911D: 50 50 50    187            HEX 5050505050505050
9125: D0 D0 D0    188            HEX D0D0D0D0D0D0D0D0
912D: 50 50 50    189            HEX 5050505050505050
9135: D0 D0 D0    190            HEX D0D0D0D0D0D0D0D0
913D: 50 50 50    191            HEX 5050505050505050
9145: D0 D0 D0    192            HEX D0D0D0D0D0D0D0D0
914D: 50 50 50    193            HEX 5050505050505050
9155: A8 A8 A8    194            HEX A8A8A8A8A8A8A8A8
915D: 28 28 28    195            HEX 2828282828282828
9165: A8 A8 A8    196            HEX A8A8A8A8A8A8A8A8
916D: 28 28 28    197            HEX 2828282828282828
9175: A8 A8 A8    198            HEX A8A8A8A8A8A8A8A8
917D: 28 28 28    199            HEX 2828282828282828
9185: A8 A8 A8    200            HEX A8A8A8A8A8A8A8A8
918D: 28 28 28    201            HEX 2828282828282828
9195: 80 80 80    202            HEX 8080808080808080
919D: 00 00 00    203            HEX 0000000000000000
91A5: 80 80 80    204            HEX 8080808080808080
91AD: 00 00 00    205            HEX 0000000000000000
91B5: 80 80 80    206            HEX 8080808080808080
91BD: 00 00 00    207            HEX 0000000000000000
91C5: 80 80 80    208            HEX 8080808080808080
91CD: 00 00 00    209            HEX 0000000000000000
                  210  ;
91D5: 3F 3B 37    211  TABHI     HEX 3F3B37332F2B2723 ;Hi-Bytes
91DD: 3F 3B 37    212            HEX 3F3B37332F2B2723
91E5: 3E 3A 36    213            HEX 3E3A36322E2A2622
91ED: 3E 3A 36    214            HEX 3E3A36322E2A2622
91F5: 3D 39 35    215            HEX 3D3935312D292521
91FD: 3D 39 35    216            HEX 3D3935312D292521
9205: 3C 38 34    217            HEX 3C3834302C282420
920D: 3C 38 34    218            HEX 3C3834302C282420
9215: 3F 3B 37    219            HEX 3F3B37332F2B2723
921D: 3F 3B 37    220            HEX 3F3B37332F2B2723
9225: 3E 3A 36    221            HEX 3E3A36322E2A2622
922D: 3E 3A 36    222            HEX 3E3A36322E2A2622
9235: 3D 39 35    223            HEX 3D3935312D292521
923D: 3D 39 35    224            HEX 3D3935312D292521
9245: 3C 38 34    225            HEX 3C3834302C282420
924D: 3C 38 34    226            HEX 3C3834302C282420
9255: 3F 3B 37    227            HEX 3F3B37332F2B2723
925D: 3F 3B 37    228            HEX 3F3B37332F2B2723
9265: 3E 3A 36    229            HEX 3E3A36322E2A2622
926D: 3E 3A 36    230            HEX 3E3A36322E2A2622
9275: 3D 39 35    231            HEX 3D3935312D292521
927D: 3D 39 35    232            HEX 3D3935312D292521
9285: 3C 38 34    233            HEX 3C3834302C282420
928D: 3C 38 34    234            HEX 3C3834302C282420
```

Zum Verstehen des Programms ist es nützlich, sich die Einzelteile mit IDUS anzusehen. Zu den Verschiebungen nach oben und unten ist wenig zu sagen, da sie „geradeaus" programmiert wurden. Die umfangreiche Kommentierung dürfte Ihnen ausreichende Hinweise geben. Die schnellen horizontalen Verschiebungen sind ähnlich einfach. Kompliziert wird es bei den Routinen RECHTS und LINKS.
Schwierigkeiten bereitet wieder einmal das Farbbit. Wenn Sie einen Punkt auf dem Bildschirm um eine Position nach rechts versetzen, muß er im Byte um eine Position nach links wandern. War er bereits im Bit 6, so darf er nicht nach Bit 7, dem Farbbit, wandern, sondern muß in das Bit 0 des nächsten Bytes gelangen. Bei einer Verschiebung nach links wiederholt sich dieses gerade andersherum: Bit 0 wandert nach Bit 6 des voranstehenden Bytes.
Bei Schwarz/Weiß-Bildern ist das Farbbit ansonsten nicht weiter problematisch. Bei einer Farbgrafik jedoch muß es erhalten bleiben. Da das Farbbit für alle 7 Punkte seines Bytes gilt, kann bei einer Verschiebung die Farbinformation nicht für alle Punkte erhalten bleiben. Zumindest 1 Bit verläßt das Byte und kommt so unter den Einfluß des benachbarten Farbbits. Die hier vorgestellte Lösung hält das Farbbit in seinem jeweiligen Byte konstant, während die Grafik-Bits wandern. Farbveränderungen sind dabei nicht auszuschließen. Müssen die Farben erhalten bleiben, so dürfen nur die FASTR- und FASTL-Routinen benutzt werden. Da diese ganze Bytes verschieben, bleibt auch das Farbbit immer bei „seinen" Grafik-Bits.
Die Programmierung von RECHTS und LINKS muß ich Ihnen noch etwas erläutern, da einige Tricks angewandt wurden, um Platz und Zeit zu sparen.
Der Kopf der Routine RECHTS arbeitet wie alle anderen. Erst ab Zeile 88 treten Besonderheiten auf. Die Speicherstelle PTR1 soll den beim Verschieben entstehenden Übertrag aufnehmen, der in das nächste Byte befördert wird. Da beim Start noch kein Übertrag vorhanden ist, wird er auf Null gesetzt (Z. 88,89). In der inneren Schleife (RL2) wird das Byte des Bildschirms geladen und mit ASL nach links geschoben. Dadurch gelangt das Farbbit in die Carry-Flagge. Anschließend wird PTR1 auf einen vorhandenen Übertrag aus dem vorangehenden Byte geprüft. Ist PTR1 negativ, so ist ein Übertrag zu berücksichtigen. Bit 0 wird durch „ORA $01" gesetzt. Das entstandene Byte wird nach PTR1 gespeichert, da sich das alte Bit 6 durch ASL jetzt in Bit 7 befindet und in der nächsten Runde das Setzen von Bit 0 bewerkstelligen soll. Unser Carry-Bit enthält immer noch die Farbinformation. Ist das Carry-Bit gelöscht, war das Farbbit 0. Mit „AND $7F" wird dieser Zustand wieder hergestellt. War das Carry-Bit gesetzt, muß Bit 7 auch gesetzt werden. Dies geschieht mit „ORA $80". Danach ist das Byte fertig und kann zurückgeschrieben werden.
Ein kleiner Trick wurde noch angewandt. Wenn in Zeile 96 das Carry-Bit gesetzt

ist, geht die Befehlsausführung in Zeile 99 weiter. So weit ist das normal. Ist aber das Carry-Bit gelöscht, wird zunächst in Zeile 97 das „AND $7F" ausgeführt. Jetzt müßten wir normalerweise das „ORA $80" überspringen, z.B. mit einem „BCC" nach Zeile 100. Stattdessen steht ein „HEX 2C" dort. Auf den ersten Blick ist das wenig sinnvoll. Auf den zweiten Blick wird das Geheimnis sichtbar: 2C 09 80 sind die drei aufeinanderfolgenden Bytes. Disassembliert ergibt dieses „BIT $8009". Der Prozessor „sieht" dadurch das „ORA $80" nicht mehr, da es im Operanden des BIT-Befehls verschwindet. Der BIT-Befehl ist an dieser Stelle sinnlos, aber ungefährlich.
Es ist eine weitverbreitete Praxis, nicht benötigte 2-Byte Befehle im Operanden von 3-Byte-Befehlen zu maskieren, z.B. um bei unterschiedlichen Einsprungadressen verschiedene Werte zu laden. Wenn Sie nur 1 Byte maskieren müssen, eignet sich „HEX 24" sehr gut dazu. Für den Prozessor liest es sich als „BIT Zeropage". Der Nachteil dieses Verfahrens wird Ihnen deutlich, wenn Sie solche Programme disassemblieren, ohne über den Quellcode zu verfügen: die Trickstellen sind nur schwer zu entschlüsseln.
Noch ein weiterer geläufiger Trick sei Ihnen verraten: Wenn Sie am Beginn einer Routine einen Einsprung benötigen, der das Carry-Bit setzt, und einen anderen Einsprung, der das Carry-Bit löscht, können sie schreiben:

```
SEC         Einsprung mit SEC
HEX 90      Ergibt BCC, wird nie als Verzweigung ausgeführt
CLC         Einsprung mit CLC
```

Der Wert 90 ergibt zusammen mit dem Opcode für CLC den Befehl BCC 18. Da vorher das Carry-Bit gerade gesetzt wurde, wird die Verzweigung nie ausgeführt, sondern nach dem vermeintlichen Distanzbyte 18, hinter dem wir das CLC verstecken, weitergemacht.

Die Routine LINKS arbeitet ganz ähnlich wie RECHTS. Da die Verschiebungsrichtung umgekehrt ist, ändert sich die Logik allerdings etwas. Hier wird zunächst das Farbbit (genaugenommen das ganze Byte) nach PTR1 gerettet. Dann wird nachgesehen, ob vom vorangehenden Byte noch ein Übertrag da ist. Falls ja (Carry gesetzt), wird Bit 7 auch gesetzt. Andernfalls wird Bit 7 gelöscht. Erst jetzt wird nach rechts geschoben (LSR). Dadurch kommt u.a. das nach Bit 7 übertragene Bit auf seinen richtigen Platz in Bit 6. Das alte Bit 0 landet im Carry-Bit und wird in der nächste Runde berücksichtigt. Zum Abschluß wird noch das alte Farbbit (aus PTR1) wieder zurückgeholt.
RECHTS und LINKS sind wunderschöne Übungsstücke für den Simulator IDUS. Am besten laden Sie vorher eine Farbgrafik nach HGR1, damit viele unterschiedliche Bytekombinationen auftauchen. Die Verschiebung der ganzen 8KByte dauert allerdings in der Simulation eine „halbe Ewigkeit".

3.7 Wir „fensterln“ ein wenig

Fenster sind heutzutage „in“. Da der Apple zu einer Zeit entstand, als noch niemand an die modernen Grafikansprüche dachte, besitzt er keine Fensterroutinen. Das heißt aber nicht, daß wir nicht selber Fenster programmieren können. Für die „Zauberkünste“ des MAC ist zwar kein Platz im Speicher des Apple II, aber für etwas bescheidenere Ansprüche tut es auch der gute alte IIer noch.
Das folgende Programm „HGR-Fenster“ ermöglicht es Ihnen, auf einer HIRES-Seite beliebig große Fenster zu öffnen. Dabei dürfen sich maximal 2 Fenster auch überlagern, ohne daß die Information im „unteren“ Fenster verloren geht. Jedes einzelne Fenster kann nach oben gescrollt werden, um auch längere Textpassagen anzeigen zu können. Das Programm ist besonders als Ergänzug zu Applesoft-Programmen gedacht, um diesen etwas mehr Pfiff zu verleihen. Es eignet sich besonders gut für Demonstrations- und Unterrichtsprogramme. Ein kleines BASIC-Programm gibt Ihnen einen Vorgeschmack, was sich mit „HGR-Fenster“ alles erreichen läßt.

Das Programm ist eigentlich ein Paket aus selbständigen Routinen, die alle einzeln aufgerufen werden können, aber zusammen für die vollständige Fensterfunktion benötigt werden. Der 1. Teil, „FENSTER“, schneidet aus der HIRES-Seite Rechtecke heraus, löscht sie und zieht an der Innenkante eine dünne Begrenzungslinie, um das Fenster besser von der Umgebung abzusetzen. Die 2. Routine ist in der Lage, Rechtecke unabhängig von ihrer Umgebung zu scrollen. Die einzige Einschränkung liegt darin, daß die Fensterbreite sich immer mit den Grenzen von ganzen Bytes decken muß. Die 3. und 4. Routine dienen schließlich dazu, in kürzester Zeit HGR1 nach HGR2 zu kopieren und umgekehrt.
Sehen Sie sich zunächst einmal das Demonstrationsprogramm an. Es wird mit „RUN HGR.W.DEMO-STARTER“ gestartet. Die Dateien „HGR. WINDOW.DEMO“, „HGR-FENSTER.OBJ“ und „HPRINTER.OBJ“ müssen sich auf der selben Diskette befinden.
Wundern Sie sich nach dem Ende der Demonstration nicht über das Verhalten Ihres Rechners. Sie sind immer noch auf der HIRES-Grafik, da das Programm „HPRINTER.OBJ“ noch aktiv ist. Wir werden es etwas später kennenlernen. Drücken Sie <RESET>, um wieder auf die normale Textseite zu kommen.

HGR.W.DEMO-STARTER

```
10  REM **** HGR - WINDOW   DEMO ****
20  REM **** (C) 1986 DR. KEHREL ****
30  HOME
40  HTAB 11: INVERSE : PRINT " HGR - WINDOW  DEMO ": NORMAL
60  PRINT  CHR$ (4)"BLOAD HGR-WINDOW.OBJ"
70  PRINT  CHR$ (4)"BLOAD HPRINTER.OBJ"
80  PRINT  CHR$ (4)"RUN HGR.WINDOW.DEMO"
```

HGR.WINDOW.DEMO

```
10  REM **** HGR - WINDOW   DEMO ****
20  REM **** (C) 1986 DR. KEHREL ****
30  HIMEM: 36864: LOMEM: 24576
40  FENSTER = 36864:SCROLL = FENSTER + 3:COPY21 = SCROLL + 3
50  FARBE = 62454:EINS = 49236:ZWEI = 49237:SEITE = 230
60  HT = 4094:VT = HT + 1: REM    $FFE,$FFF
70  KOPY12 = COPY21 + 3:YANF = 251:XANF = 252:XEN = 253:
    HOEHE = 254
80  HGR2 : CALL 3819: REM    $EEB
90  POKE VT,0: POKE HT,7
100  PRINT "Wir schreiben jetzt auf HGR2"
105  HCOLOR= 3: HPLOT 0,191 TO 40,50 TO 80,191 TO 120,50 TO
     160,191 TO 200,50 TO 240,191 TO 279,50
110  GOSUB 2000
120  HPLOT 0,0: CALL FARBE
130  GOSUB 2100: GOSUB 2000
140  CALL FENSTER: POKE 32,6: POKE 33,23: POKE 34,4: POKE 35,8
150  POKE HT,6: POKE VT,4: PRINT "Dieses ist ein Fenster.":
     POKE HT,6
160  PRINT "Wir schreiben Text hin-": POKE HT,6
170  PRINT "ein, bis es voll ist.": GOSUB 2000: CALL SCROLL
180  POKE HT,6: PRINT "Dann rollen wir einfach";:
     GOSUB 2000: CALL SCROLL
190  POKE HT,6: PRINT "den Ausschnitt hoch!";: GOSUB 2000:
     CALL SCROLL: CALL SCROLL
200  CALL COPY21: POKE HT,6: PRINT "Es gibt sehr ...";: POKE
     YANF,100: POKE XANF,5: POKE XEN,13: POKE HOEHE,14
210  GOSUB 2000: CALL FENSTER: POKE VT,13: POKE HT,6:
     PRINT "kleine";
220  POKE YANF,80: POKE XANF,14: POKE XEN,39: POKE HOEHE,100
230  CALL FENSTER: POKE HT,15: PRINT "und sehr grosse Fenster"
240  POKE HT,15: GOSUB 2000: PRINT "Mit einem kleinen Trick"
250  POKE HT,15: PRINT "verschwinden sie wieder": GOSUB 2000
260  CALL FENSTER: GOSUB 2000: CALL KOPY12: GOSUB 2000:
     GOSUB 2100: CALL FENSTER
270  HPLOT 0,0: CALL FARBE: GOSUB 2000: HCOLOR= 2: HPLOT 0,0:
     CALL FARBE
280  GOSUB 2000: CALL FENSTER: POKE VT,4: POKE HT,6: PRINT
     "Fuer kleine Demonstra-"
290  POKE HT,6: PRINT "tionen ist das voellig": POKE HT,6:
     PRINT "ausreichend."
300  CALL COPY21: POKE YANF,18: POKE XANF,20: POKE XEN,39:
     POKE HOEHE,58
```

```
310  GOSUB 2000: CALL FENSTER: POKE VT,3: POKE HT,21: PRINT "Sie
     koennen auch"
320  POKE HT,23: PRINT "zwei Fenster": POKE HT,23:
     PRINT "ueberlagern!"
330  GOSUB 2000: CALL KOPY12: GOSUB 2000: GOSUB 2100:
     CALL FENSTER: GOSUB 2000
340  POKE HT,14: POKE VT,6: PRINT "E N D E"
350  END
2000  FOR I = 1 TO 2000: NEXT I: RETURN : REM    PAUSE
2100  POKE YANF,28: POKE XANF,5: POKE XEN,30: POKE
      HOEHE,38: RETURN
```

Im BASIC-Programm heißt es „KOPY12“ und „COPY21“, da Applesoft nur die ersten zwei Zeichen beachtet und „COPY12“ für Applesoft die gleiche Variable wie „COPY21“ ist.

HGR-WINDOW

```
1     ;*******************************
2     ;          HGR - FENSTER        *
3     ; (C) 1985   Dr. Jürgen Kehrel *
4     ;*******************************
5     ;
6     HGR1       EQU $0006
7     PAGE1      EQU $0007
8     HGR2       EQU $0008
9     PAGE2      EQU $0009
10    WNDLFT     EQU $0020
11    WNDWDTH    EQU $0021
12    WNDTOP     EQU $0022
13    WNDBTM     EQU $0023
14    BASL       EQU $004C
15    BASH       EQU $004D
16    BAS2L      EQU $002A
17    BAS2H      EQU $002B
18    HPAGE      EQU $00E6
19    BASEADR    EQU $00EB       ;+$00EC
20    XHIRE      EQU $00ED
21    XHILI      EQU $00EE
22    YENDE      EQU $00FA       ;0 - 191
23    YSTART     EQU $00FB       ;0 - 191
24    XSTART     EQU $00FC       ;0 - 39 !!
25    XENDE      EQU $00FD       ;0 - 39 !!
26    YHOEHE     EQU $00FE       ;0 - 191
27    ;
28    HPLOT      EQU $F457
29    HGLIN      EQU $F53A
30    SETHCOL    EQU $F6F0
31    ;
32    ;
33               ORG $9000
34    ;
35    ; Sprünge zu den einzelnen Routinen
36    ;
```

```
9000: 4C 0C 90  37            JMP FENSTER
9003: 4C B4 90  38            JMP HISCROLL
9006: 4C 2E 91  39            JMP COPY21
9009: 4C 4E 91  40            JMP COPY12
                41  ;
                42  ;
                43  ; Modifiziert die Adressenberechnung
                44  ; je nach aktueller HGR-Seite ($E6)
                45  ;
900C: A5 E6     46  FENSTER   LDA HPAGE       ;$20 oder $40
900E: 4A        47            LSR
900F: 4A        48            LSR             ;$08 oder $10
9010: 8D A0 90  49            STA SEITE+1
9013: D0 01     50            BNE FNS
9015: 60        51            RTS             ;ungültig
                52  ;
                53  ; Zum Beginn des Fensters (oben) die
                54  ; Fensterhöhe addieren und als End-
                55  ; wert des Fensters merken. Muß inner-
                56  ; halb des Bildschirms bleiben (<192).
                57  ;
9016: A5 FB     58  FNS         LDA YSTART
9018: AA        59              TAX             ;retten
9019: 38        60              SEC
901A: 65 FE     61              ADC YHOEHE
901C: C9 C0     62              CMP #$C0        ;< 192 ??
901E: B0 79     63              BGE FNENDE
9020: 85 FA     64              STA YENDE
                65  ;
                66  ; In der Zeile X die erforderliche
                67  ; Breite löschen (auf $00 setzen).
                68  ;
9022: 8A        69  CLRZEILE    TXA
9023: 20 9A 90  70              JSR CALCADR
9026: A4 FC     71              LDY XSTART
9028: A9 00     72              LDA #$00
902A: 91 EB     73  CLRLOOP    STA (BASEADR),Y
902C: C8        74              INY
902D: C4 FD     75              CPY XENDE
902F: 90 F9     76              BLT CLRLOOP
                77  ;
                78  ; Nächste Zeile löschen bis ENDE
                79  ;
9031: E8        80              INX
9032: E4 FA     81              CPX YENDE
9034: 90 EC     82              BLT CLRZEILE
                83  ;
                84  ; Farbe „Weiß" auswählen
                85  ;
9036: A2 03     86              LDX #$03        ;Weiß
9038: 20 F0 F6  87              JSR SETHCOL
                88  ;
```

```
                        89   ; Aus der X Byte-Position die Bit-
                        90   ; Position für den Rahmen berechnen.
                        91   ; X = ( XSTART * 7) + 1
                        92   ;
903B: A0 00             93            LDY #$00        ;Hi-Byte X
903D: A5 FC             94            LDA XSTART      ;Lo-Byte X
903F: 0A                95            ASL
9040: 0A                96            ASL
9041: 0A                97            ASL             ;* 8
9042: 90 01             98            BCC NOINC       ;Übertrag?
9044: C8                99            INY             ;Ja
9045: 38                100  NOINC     SEC             ;1* abziehen
9046: E5 FC             101           SBC XSTART      ;= * 7
9048: B0 01             102           BCS NOINC1      ;Übertrag?
904A: 88                103           DEY             ;Ja, abziehen
904B: 18                104  NOINC1    CLC
904C: 69 01             105           ADC #$01        ;+ 1
904E: 90 01             106           BCC NOINC2      ;Übertrag?
9050: C8                107           INY             ;Ja
9051: AA                108  NOINC2    TAX             ;Lo-Byte X
9052: 48                109           PHA             ;merken
9053: 84 EE             110           STY XHILI
                        111  ;
                        112  ; Y-Wert des Rahmens festlegen
                        113  ;
9055: A5 FB             114           LDA YSTART
9057: 18                115           CLC
9058: 69 01             116           ADC #$01
                        117  ;
                        118  ; 1 Punkt oben links setzen
                        119  ;
905A: 20 57 F4          120           JSR HPLOT
                        121  ;
                        122  ; Horiz. Linie nach rechts ziehen
                        123  ;
905D: A4 FB             124           LDY YSTART
905F: C8                125           INY
9060: A2 00             126           LDX #$00        ;X-Hi
9062: A5 FD             127           LDA XENDE       ;X-Lo
9064: 0A                128           ASL
9065: 0A                129           ASL
9066: 0A                130           ASL             ;* 8
9067: 90 01             131           BCC NOINC3      ;Übertrag?
9069: E8                132           INX             ;Ja
906A: 38                133  NOINC3    SEC             ;1* abziehen
906B: E5 FD             134           SBC XENDE       ;= * 7
906D: 08                135           PHP             ;Status retten
906E: 38                136           SEC
906F: E9 02             137           SBC #$02        ; - 2
9071: 28                138           PLP             ;Status zurück
9072: B0 01             139           BCS NOINC4      ;Übertrag?
9074: CA                140           DEX             ;Ja
9075: 48                141  NOINC4    PHA             ;merken
9076: 86 ED             142           STX XHIRE
```

```
9078: 20 3A F5  143             JSR HGLIN
                144  ;
                145  ; Vert. Linie nach unten
                146  ;
907B: 68        147             PLA             ;X-Lo
907C: A6 ED     148             LDX XHIRE
907E: A4 FA     149             LDY YENDE
9080: 88        150             DEY
9081: 88        151             DEY             ; - 2
9082: 20 3A F5  152             JSR HGLIN
                153  ;
                154  ; Horz. Linie nach links
                155  ;
9085: A4 FA     156             LDY YENDE
9087: 88        157             DEY
9088: 88        158             DEY             ; - 2
9089: A6 EE     159             LDX XHILI
908B: 68        160             PLA             ;X- Lo
908C: 48        161             PHA             ;und zurück
908D: 20 3A F5  162             JSR HGLIN
                163  ;
                164  ; Kästchen schließen durch verti-
                165  ; kale Linie nach oben zum Anfang
                166  ;
9090: A6 EE     167             LDX XHILI
9092: 68        168             PLA
9093: A4 FB     169             LDY YSTART
9095: C8        170             INY
9096: 20 3A F5  171             JSR HGLIN
                172  ;
                173  ; Wir sind fertig
                174  ;
9099: 60        175  FNENDE     RTS
                176  ;
                177  ; Adresse einer Zeile berechnen
                178  ; IN: Akku enthält Zeilennummer (0-191)
                179  ; OUT: (BASEADR) enthält gesuchte Adresse
                180  ;
909A: A8        181  CALCADR    TAY
909B: 29 C7     182             AND #$C7
909D: 85 EB     183             STA BASEADR
909F: 09 08     184  SEITE      ORA #$08        ;wird „gepoked"
90A1: 85 EC     185             STA BASEADR+1
90A3: 98        186             TYA
90A4: 0A        187             ASL
90A5: 0A        188             ASL
90A6: 66 EB     189             ROR BASEADR
90A8: 0A        190             ASL
90A9: 26 EC     191             ROL BASEADR+1
90AB: 66 EB     192             ROR BASEADR
90AD: 0A        193             ASL
90AE: 26 EC     194             ROL BASEADR+1
90B0: 0A        195             ASL
90B1: 66 EB     196             ROR BASEADR
90B3: 60        197             RTS
```

```
                    198  ;
                    199  ;
                    200  ;*******************
                    201  ;  HIRES - SCROLL  *
                    202  ;*******************
                    203  ;
                    204  ; Scrollt ein HIRES-Fenster nach oben
                    205  ; Übernimmt Fensterparameter vom nor-
                    206  ; malen Textbildschirm. Es sind immer
                    207  ; nur Blöcke von 8 * 1 Byte schiebbar.
                    208  ;
90B4: A5 22         209  HISCROLL  LDA WNDTOP
90B6: 48            210           PHA
90B7: 20 11 91      211           JSR ADRESSE
                    212  ;
                    213  ; Alle „Zeilen“ im Fenster hoch
                    214  ; Eine „Zeile“ besteht aus 8 unter-
                    215  ; einanderliegenden Bytes (HGR-Zeilen)
                    216  ;
90BA: A5 2A         217  LOOP      LDA BAS2L       ;Zeile „jetzt“
90BC: 85 4C         218           STA BASL        ;wird Zielzeile
90BE: A5 2B         219           LDA BAS2H
90C0: 29 E3         220           AND #$E3        ;%11100011
90C2: 85 4D         221           STA BASH
90C4: 68            222           PLA             ;alte Zeile zurück
90C5: 18            223           CLC
90C6: 69 01         224           ADC #$01        ;+ 1
90C8: C5 23         225           CMP WNDBTM      ;schon unten?
90CA: B0 22         226           BGE CLEAR       ;Ja, letzte löschen
90CC: 48            227           PHA             ;merken
90CD: 20 11 91      228           JSR ADRESSE     ;Basisadresse
                    229  ;
90D0: A2 07         230           LDX #$07        ; 8 Bytes
90D2: A4 21         231  L0        LDY WNDWDTH
90D4: 88            232           DEY
90D5: B1 2A         233  L1        LDA (BAS2L),Y
90D7: 91 4C         234           STA (BASL),Y
90D9: 88            235           DEY
90DA: 10 F9         236           BPL L1
                    237  ;
90DC: CA            238           DEX
90DD: 30 DB         239           BMI LOOP
                    240  ;
                    241  ; Die nächste HGR-Zeile liegt um
                    242  ; $0400 Bytes höher im Speicher
                    243  ;
90DF: 18            244           CLC
90E0: A5 2B         245           LDA BAS2H
90E2: 69 04         246           ADC #$04
90E4: 85 2B         247           STA BAS2H
90E6: A5 4D         248           LDA BASH
90E8: 69 04         249           ADC #$04
90EA: 85 4D         250           STA BASH
90EC: D0 E4         251           BNE L0          ;immer
                    252  ;
```

```
                      253  ; unterste „Zeile" auf Schwarz löschen
                      254  ;
90EE: A5 23           255  CLEAR     LDA WNDBTM
90F0: E9 01           256            SBC #$01
90F2: A0 00           257            LDY #$00
90F4: 20 11 91        258            JSR ADRESSE
90F7: A2 07           259            LDX #$07        ;8 *
90F9: A9 00           260  CL0       LDA #$00
90FB: 91 2A           261  CL1       STA (BAS2L),Y
90FD: C8              262            INY
90FE: C4 21           263            CPY WNDWDTH
9100: 90 F9           264            BLT CL1
9102: CA              265            DEX
9103: 30 0B           266            BMI CLRTS
                      267  ;
                      268  ; Adresse der nächsten HGR-Zeile
                      269  ; liegt um $0400 höher im Speicher
                      270  ;
9105: A0 00           271            LDY #$00
9107: A5 2B           272            LDA BAS2H
9109: 18              273            CLC
910A: 69 04           274            ADC #$04
910C: 85 2B           275            STA BAS2H
910E: D0 E9           276            BNE CL0         ;IMMER
                      277  ;
9110: 60              278  CLRTS     RTS             ;Fertig
                      279  ;
                      280  ; Berechnet die HGR-Adresse
                      281  ; IN: Akku mit Zeilennummer (0 - 23)
                      282  ; OUT : (BAS2L) Basisadresse mit
                      283  ; berücksichtigtem linken Rand
                      284  ;
9111: 4A              285  ADRESSE   LSR
9112: AA              286            TAX
9113: 29 03           287            AND #$03
9115: 05 E6           288            ORA HPAGE       ;HGR-SEITE
9117: 85 2B           289            STA BAS2H
9119: BD 22 91        290            LDA TAB,X
911C: 6A              291            ROR
911D: 65 20           292            ADC WNDLFT
911F: 85 2A           293            STA BAS2L
9121: 60              294            RTS
                      295  ;
                      296  ;
9122: 00 00 00        297  TAB       HEX 00000000
9126: 50 50 50        298            HEX 50505050
912A: A0 A0 A0        299            HEX A0A0A0A0
                      300  ;
                      301  ;*******************************
                      302  ;  Kopiert HGR2 nach HGR 1     *
                      303  ;*******************************
                      304  ;
912E: A9 20           305  COPY21    LDA #$20        ;$2000
9130: 85 07           306            STA PAGE1
```

```
9132: 0A       307            ASL             ;$4000
9133: 85 09    308            STA PAGE2
9135: A9 00    309            LDA #$00
9137: 85 06    310            STA HGR1
9139: 85 08    311            STA HGR2
913B: A8       312            TAY             ;Y=0
913C: B1 08    313  CLR       LDA (HGR2),Y
913E: 91 06    314            STA (HGR1),Y
9140: C8       315            INY
9141: D0 F9    316            BNE CLR
9143: E6 07    317            INC PAGE1
9145: E6 09    318            INC PAGE2       ;weitersetzen
9147: A5 09    319            LDA PAGE2       ;Ende testen
9149: C9 60    320            CMP #$60
914B: 90 EF    321            BLT CLR         ;weiter
914D: 60       322            RTS
               323  ;
               324  ;
               325  ;*******************************
               326  ;  Kopiert HGR 1 nach HGR 2    *
               327  ;*******************************
               328  ;
914E: A9 20    329  COPY12     LDA #$20        ;$2000
9150: 85 07    330            STA PAGE1
9152: 0A       331            ASL             ;$4000
9153: 85 09    332            STA PAGE2
9155: A9 00    333            LDA #$00
9157: 85 06    334            STA HGR1
9159: 85 08    335            STA HGR2
915B: A8       336            TAY             ;Y=0
915C: B1 06    337  CLR1      LDA (HGR1),Y
915E: 91 08    338            STA (HGR2),Y
9160: C8       339            INY
9161: D0 F9    340            BNE CLR1
9163: E6 07    341            INC PAGE1
9165: E6 09    342            INC PAGE2       ;weitersetzen
9167: A5 09    343            LDA PAGE2       ;Ende testen
9169: C9 60    344            CMP #$60
916B: 90 EF    345            BLT CLR1        ;weiter
916D: 60       346            RTS
```

Die beiden Kopierroutinen sind recht einfach. Es werden jeweils die Bytes der Zero-Page auf den Beginn des Bildschirmspeichers ($2000 bzw. $4000) gesetzt und dann in einer kleinen inneren Schleife, mit den Y-Register als Laufvariablen, und in einer großen Außenschleife solange Bytes kopiert (LDA/STA), bis wir bei der Adresse $6000 angekommen und damit fertig sind.
Die Routine „FENSTER“ ist schon schwieriger und auch recht lang. Vielleicht schreiben Sie eine kürzere und elegantere Lösung. Da der Aufruf auch von BASIC aus geschehen soll, brauchen wir eine Schnittstelle zwischen den Programmen. Die Speicherstellen 251 bis 254 ($00FB bis $00FE) dienen uns dazu. 255 wurde nicht gewählt, da diese Speicherstelle bei jedem STR$-Befehl vom

Applesoft-Interpreter benutzt und überschrieben wird. In vielen Büchern ist $00FF trotzdem fälschlich als frei angegeben.

$00FA = 250 = YENDE: Y-Wert der Untergrenze (wird berechnet)
$00FB = 251 = YSTART: Y-Wert der Obergrenze (0 - 191)
$00FC = 252 = XSTART: X-Wert der linken Grenze (0 - 39)
$00FD = 253 = XENDE: X-Wert der rechten Grenze (0 - 39)
$00FE = 254 = YHOEHE: Höhe des Fensters (0 - 191)

Der Wert YENDE wird aus YSTART und YHOEHE berechnet, die übrigen vier Werte müssen vom BASIC-Programm gesetzt werden.
Da das Programm sowohl mit HGR1 als auch mit HGR2 laufen soll, wird die Information über die aktive (gewünschte) Seite aus der Speicherstelle $00E6 (HPAGE) gewonnen und zur Modifikation der Adress-Berechnungsroutine benutzt. CRLZEILE löscht eine Zeile im Fenster, indem die betroffenen Bytes auf $00 besetzt werden. Interessant ist daran die Adressberechnung in CALCADR. CALCADR berechnet zu der im Akku übergebenen Zeilennummer die Basisadresse und legt sie in (BASEADR) ab. Auch hier lohnt sich eine Untersuchung mit IDUS. In Zeile 184 wird entschieden, um welche HGR-Seite es sich handelt. „ORA $08" berechnet Adressen von HGR1, „ORA $10" solche von HGR2. Diese Routine ist kürzer und schneller als die ROM-Routine des Applesoft. An die Geschwindigkeit von Tabellen wie im Programm „HI-RES-SCROLL" kommt aber auch sie nicht heran.

CALCADR gehört zu jenen Routinen, die auf Anhieb nicht zu durchschauen sind. Sie entstand nach und nach, wurde immer weiter optimiert, bis sie die heutige Form fand. Um sie zu verstehen, müssen wir uns noch einmal mit der Organisation der HIRES-Seite beschäftigen.
Die HIRES-Seite läßt sich in 3 Blöcke einteilen. Jeder Block besteht aus 8 Gruppen, jede Gruppe aus 8 Zeilen. Diese Information läßt sich in den Bits der Zeilennummern wiederfinden. Bit 6 und 7 geben den Block an, Bit 3 - 5 die Gruppe und Bit 0 - 2 die Zeile. Wir teilen deshalb das Byte der Zeilennummer wie folgt auf:

```
B B  G G G  Z Z Z
1 2  1 2 3  1 2 3
```

B steht für Block, G für Gruppe und Z für Zeile. Um jetzt zur Grundadresse der zugehörigen Grafikzeile zu kommen, müssen wir diese Bits ein wenig umsortieren. Für das Hi- und das Lo-Byte ergeben sich folgende Verhältnisse:

```
     Hi-Byte              Lo-Byte
- P P Z Z Z G G     G B B B B - - -
-     1 2 3 1 2     3 1 2 1 2 - - -
```

Bit 5 und 6 vom Hi-Byte (PP) enthalten die Information über die HGR-Seite. 10 steht für Seite 2, ein 01 für Seite 1. Statt der Striche ist das betreffende Bit auf Null zu setzen.

Versuchen wir als Beispiel die Zeile $55. Wir finden folgende Aufteilung:

```
B B  G G G  Z Z Z
1 2  1 2 3  1 2 3
=================
0 1  0 1 0  1 0 1   = $ 55
```

Wenn wir die Adresse für HGR1 ausrechnen wollen, steht 01 für PP.

```
     Hi-Byte              Lo-Byte
- P P Z Z Z G G     G B B B B - - -
-     1 2 3 1 2     3 1 2 1 2 - - -
=================  =================
0 0 1 1 0 1 0 1     0 0 1 0 1 0 0 0
      $35                 $28
```

Als Adresse ergibt sich demnach $3528.
Die Routine CALCADR besorgt nun nichts anderes, als die Bits entsprechend dem obigen Schema umzuverteilen. Neben IDUS helfen hier auch noch Blatt und Bleistift weiter, wenn Sie in der Simulation verfolgen wollen, wo welches Bit bleibt.

Beenden wir unseren Exkurs. Wenn der Fensterbereich gelöscht ist, wird an der Innenseite eine weiße Linie gezogen. Dazu wird die uns schon geläufige Routine HGLIN benutzt. Da die rechten und linken Fenstergrenzen in „Bytes“ und nicht in „Spalten“ angegeben wurden, muß zunächst der Bytewert in die Spalte umgerechnet werden, indem der Bytewert mit 7 multipliziert wird. Da es eine einfache Multiplikation mit 7 nicht gibt, wird mit 8 multipliziert, was durch einfaches dreimaliges Linksschieben zu bewerkstelligen ist. Anschließend wird der Bytewert einmal abgezogen, so daß das Ergebnis eine Multiplikation mit 7 ist.
Da die Linien innen liegen sollen, sind noch einige kleine Korrekturen nötig. Der Rest ist einfach.

HISCROLL besitzt eine eigene Schnittstelle zum BASIC. Die Fenstervariablen der Textseite dienen zur Übergabe des Scrollfensters.

$0020 = 32 = WNDLFT = Linker Rand (0 - 39)
$0021 = 33 = WNDWDTH = Fensterbreite (0 - 39)
$0022 = 34 = WNDTOP = Oberer Rand (0 - 23)
$0023 = 35 = WNDBTM = Unterer Rand (0 - 23)

Dieser Programmteil benutzt eine alternative Form der Adressbestimmung. Die Routine ADRESSE berechnet die Adresse der Grundlinie einer Gruppe, wenn ihr die Nummer der Gruppe (vergleichbar mit der Textzeilennummer) im Akkumulator übergeben wird. Hier wird wieder eine kleine Tabelle benutzt. Die folgenden 7 Zeilen werden durch Addition von $400 zur vorangehenden Adresse erhalten.

3.8 Zeichnen, ohne gesehen zu werden

Die HIRES-ROM-Routinen des Apple sind sowohl für HGR1 als auch für HGR2 ausgelegt. Welche Seite zum Plotten benutzt wird, legt die Speicherstelle HPAG ($00E6) fest. Der Wert $20 führt dazu, daß HGR1 benutzt wird, ein $40 gehört zu HGR2. Wenn Sie $60 eintragen, wird der Speicher von $6000 bis $7FFF benutzt. Dieser Speicherbereich wird deshalb von manchen als HGR3 bezeichnet. Wenn Sie kein DOS im Rechner haben oder ein in die Language-Card ge„move"tes DOS 3.3 benutzen, können Sie mit $80 in HPAG noch den Speicher von $8000 bis $9FFF als HGR4 nutzen. Sowohl HGR3 als auch HGR4 können von der Elektronik des Apple nicht angezeigt werden. Sie müssen deshalb die beiden imaginären Seiten HGR3 und HGR4 nach dem Zeichnen in eine der beiden Seiten HGR1 oder HGR2 kopieren.
Welche Seite angezeigt wird, hängt nur von der Stellung der Softswitches ab. Daraus folgt, das Zeichnen und Anzeigen zwei unabhängige Prozesse sind. Sie können also z.B. HGR1 anzeigen und gleichzeitig auf HGR2 (oder HGR3 oder HGR4) zeichnen.
Werden große Objekte auf dem HIRES-Schirm bewegt, entsteht oft ein augenbelastendes Flimmern, da es eine merklich Zeit braucht, bis das Objekt an seinem alten Ort gelöcht und etwas versetzt am neuen wieder aufgebaut ist. Die Ausnutzung der verschiedenen HGR-Seiten (Page-Flipping) ermöglicht eine weitgehende Beruhigung der Anzeige.

Der Ablauf vollzieht sich dabei nach folgenden Prinzip:
1. Das Objekt wird auf Seite 1 gezeichnet und angezeigt.
2. HPAG wird auf $40 gesetzt und das Objekt (unsichtbar) an seiner neuen Position gezeichnet.
3. Mit dem Softswitch HISCR wird HGR2 zur Anzeige gebracht.
4. HPAG wird wieder auf $20 gesetzt, der Bildschirm (unsichtbar) gelöscht und das Objekt wieder etwas weiter versetzt gezeichnet.
5. Mit dem Softswitch LOWSCR wird HGR1 zur Anzeige gebracht.

Die Schritte 2 bis 5 wiederholen sich jetzt laufend, bis das Objekt sich nicht mehr bewegt. Da das Löschen und Neuzeichnen nicht sichtbar und der Seitenwechsel für den Betrachter nicht erkennbar ist, ergibt sich eine ruhige Bewegung.

Wenn Sie ein Objekt vor einem Hintergrund bewegen wollen, können Sie diesen Hintergrund ohne das Objekt in HGR3 einmal aufbauen und dann unverändert dort belassen. HGR3 wird mit einer Routine, die wie COPY21 aufgebaut ist, jeweils in die nicht sichtbare Seite kopiert. Dadurch wird gleichzeitig der Hintergrund neu aufgebaut *und* das alte Objekt gelöscht. Dieser Vorgang dauert nicht länger als das reine Löschen einer HIRES-Seite.

Wenn Sie mehr über Grafikanimation wissen möchten, finden Sie in „Bewegte Apple-Grafik" aus dem Hüthig-Verlag oder in „Apple II Raster Grafik" von Pandabooks (jetzt Ampersand) viele Anregungen für Assembler-Programmierer.

Zum Abschluß noch eine Warnung: die HIRES-Routinen produzieren mit jedem beliebigen Wert in HPAG Zeichnungen im Speicher, nur nicht immer dort, wo Sie es gerne hätten. Benutzen Sie also nur die Werte $20, $40, $60 und gegebenenfalls $80.

3.9 Punkt, Punkt, Komma, Strich

Bei der Fenster-Demonstration haben wir schon Schrift auf der HIRES-Seite erzeugt. Bislang habe ich Ihnen aber noch nicht verraten, wie dies programmiert wurde.
Um Buchstaben zu erzeugen, müssen wir die Grafikpunkte des HIRES-Schirms so anordnen, daß sie das Schriftzeichen erkennen lassen. Um die gleiche

Schriftgröße wie im Textmodus zu erzielen, muß ein Zeichen bis zu 8 Punkten hoch sein. Seine Breite darf 1 Byte betragen, also 7 Punkte. Wir sprechen deshalb von einer 7*8 Matrix. Damit benachbarte Zeichen nicht aneinanderstoßen, muß auf mindestens einer Seite, möglichst aber auf beiden ein Punkt frei bleiben. Die nutzbare Matrix schränkt sich dadurch auf 5*8 Punkte ein. Da einige Zeichen Unterlängen besitzen, muß auch nach unten in den meisten Fällen ein Punkt frei bleiben.
Um ein Zeichen zu entwerfen, benutzen Sie am besten kariertes Papier. Wir wollen jetzt zusammen ein großes A darstellen.

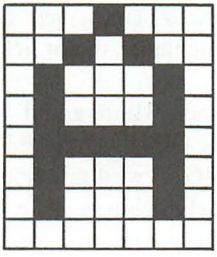


Abb. 6: Rasterdarstellung von „A"

Da das A symmetrisch ist, läßt es sich sehr einfach codieren. Die Spiegelung entfällt hier, und wenn wir das Farbbit auf Null setzen, ergeben sich für die Zeilen folgende Byte-Werte:

% 0 000 1000 = $08
% 0 001 0100 = $14
% 0 010 0010 = $22
% 0 010 0010 = $22
% 0 011 1110 = $3E
% 0 010 0010 = $22
% 0 010 0010 = $22
% 0 000 0000 = $00

Da es sich bei Buchstaben zumeist um statische Objekte handelt, die nicht auf dem Bildschirm hin- und herbewegt werden, können wir eine weitere Eigenschaft der Applegrafik ausnutzen. Wenn das Farbbit gesetzt ist, ändert sich nicht nur die Farbe der Punkte, sie werden auch ein kleines Stück (½ Punktbreite) nach rechts verschoben. Wir nutzen diese Tatsache, um einige Buchstaben etwas abgerundeter erscheinen zu lassen. Das „A" läßt sich wie folgt codieren:

% 0 000 1000 = $08
% 0 001 0100 = $14
% 1 001 0010 = $92
% 0 010 0010 = $22
% 0 011 1110 = $3E
% 0 010 0010 = $22
% 0 010 0010 = $22
% 0 000 0000 = $00

Bitte denken Sie daran, daß auf dem Bildschirm die Bits spiegelbildlich erscheinen!

Am Ende des Programms „HIRES-Schriften-Generator" finden Sie einen vollständigen Zeichensatz, der auf diese Weise zustande gekommen ist. Auf der Begleitdiskette ist er zusätzlich noch als separater File „CHAR.SET" enthalten. Alle Fonts der „DOS Programmers Toolkit"-Diskette von Apple können Sie ebenso benutzen, da sie vom inneren Aufbau her gleich sind.

Der „HIRES-Schriften-Generator" enthält in seinem vorderen Teil ein Demonstrationsprogramm, das Ihnen die Benutzung des Generators zeigt. Es stammt aus einer Testversion von IDUS, wie sie leicht erkennen können. Anschließend werden alle Ausgaben zum Bildschirm über den Generator auf die HIRES-Seite umgelenkt werden. Die erlaubten Befehle sind im Listing (Zeile 188ff) aufgeführt.

HPRINTER

```
1    ;*******************************
2    ;  HIRES-Schriften-Generator   *
3    ; (C) 1985  Dr. Jürgen Kehrel  *
4    ;*******************************
5    ;
6              ORG $D78        ;Tabelle ab $1000
7    ;
8    ADR       EQU $0000       ;+ $0001
9    HGRADR    EQU $0002       ;+ $0003
10   PTR       EQU $0004       ;+ $0005
11   CSWL      EQU $0036
12   CSWH      EQU $0037
13   HPAGE     EQU $00E6
14   ;
15   CONNECT   EQU $03EA
16   ;
17   KBD       EQU $C000
18   STROBE    EQU $C010
19   ;
20   WAIT      EQU $FCA8
21   HGR2      EQU $F3D8
```

```
                      22   ;
                      23   ;
                      24   ;
0D78: D8              25   START     CLD             ;Binärmodus
0D79: 20 D8 F3        26             JSR HGR2        ;HGR2 initial.
                      27   ;
                      28   ; Rahmen zeichnen
                      29   ;
0D7C: A0 00           30             LDY #$00
0D7E: 8C FE 0F        31             STY HORZ
0D81: 8C FF 0F        32             STY VERT
0D84: A9 AA           33             LDA #"*"
0D86: 20 F6 0E        34             JSR STERN
0D89: 8C FE 0F        35             STY HORZ
0D8C: A2 17           36             LDX #23
0D8E: 8E FF 0F        37             STX VERT
0D91: 20 F6 0E        38             JSR STERN
0D94: EE FF 0F        39             INC VERT
0D97: A2 27           40   LINIE     LDX #39
0D99: 8E FE 0F        41             STX HORZ
0D9C: 20 2E 0F        42             JSR HPRINT
0D9F: 20 2E 0F        43             JSR HPRINT
0DA2: AE FF 0F        44             LDX VERT
0DA5: D0 F0           45             BNE LINIE
                      46   ;
                      47   ; Titel ausgeben
                      48   ;
0DA7: A9 0E           49             LDA #14
0DA9: 8D FE 0F        50             STA HORZ
0DAC: A9 05           51             LDA #5
0DAE: 8D FF 0F        52             STA VERT
0DB1: CE DA 0F        53             DEC INV+1       ;Inverse
0DB4: A9 C9           54             LDA #"I"
0DB6: 20 2E 0F        55             JSR HPRINT
0DB9: EE FF 0F        56             INC VERT
0DBC: A9 C4           57             LDA #"D"
0DBE: 20 2E 0F        58             JSR HPRINT
0DC1: EE FF 0F        59             INC VERT
0DC4: A9 D5           60             LDA #"U"
0DC6: 20 2E 0F        61             JSR HPRINT
0DC9: EE FF 0F        62             INC VERT
0DCC: A9 D3           63             LDA #"S"
0DCE: 20 2E 0F        64             JSR HPRINT
0DD1: EE DA 0F        65             INC INV+1       ;Normal
0DD4: A2 08           66             LDX #8
0DD6: A0 02           67             LDY #2
0DD8: 20 FF 0E        68             JSR STRINGPR
0DDB: C8 F5 E5        69             ASC "Huethig Software Service"
0DF3: 00              70             HEX 00
0DF4: A2 0F           71             LDX #15
0DF6: A0 05           72             LDY #5
0DF8: 20 FF 0E        73             JSR STRINGPR
0DFB: CE D4 C5        74             ASC "NTERAKTIVER"
0E06: 00              75             HEX 00
```

```
0E07: E8          76              INX
0E08: C8          77              INY
0E09: 20 FF 0E    78              JSR STRINGPR
0E0C: C5 C2 D5    79              ASC "EBUGGER"
0E13: 00          80              HEX 00
0E14: E8          81              INX
0E15: C8          82              INY
0E16: 20 FF 0E    83              JSR STRINGPR
0E19: CE C4       84              ASC "ND"
0E1B: 00          85              HEX 00
0E1C: E8          86              INX
0E1D: C8          87              INY
0E1E: 20 FF 0E    88              JSR STRINGPR
0E21: C9 CD D5    89              ASC "IMULATOR"
0E29: 00          90              HEX 00
0E2A: A2 0D       91              LDX #13
0E2C: A0 0A       92              LDY #10
0E2E: 20 FF 0E    93              JSR STRINGPR
0E31: B6 B5 B0    94              ASC "6502 Prozessor"
0E3F: 00          95              HEX 00
0E40: A2 07       96              LDX #7
0E42: A0 0D       97              LDY #13
0E44: 20 FF 0E    98              JSR STRINGPR
0E47: F6 EF EE    99              ASC "von  Dr. Juergen B. Kehrel"
0E61: 00          100             HEX 00
0E62: A2 11       101             LDX #17
0E64: A0 0F       102             LDY #15
0E66: 20 FF 0E    103             JSR STRINGPR
0E69: A8 B1 B9    104             ASC "(1985)"
0E6F: 00          105             HEX 00
0E70: A2 06       106             LDX #6
0E72: A0 15       107             LDY #21
0E74: 20 FF 0E    108             JSR STRINGPR
0E77: A8 E3 A9    109             ASC "(c) Dr. Alfred Huethig Verlag"
0E94: 00          110             HEX 00
0E95: A2 0F       111             LDX #15
0E97: C8          112             INY
0E98: 20 FF 0E    113             JSR STRINGPR
0E9B: C8 E5 E9    114             ASC "Heidelberg"
0EA5: 00          115             HEX 00
                  116 ;
                  117 ; Pause einlegen, bei Tasten-
                  118 ; druck „sofort" weiter
                  119 ;
0EA6: A2 45       120             LDX #$45        ;Pause
0EA8: A9 C4       121 WARTEN      LDA #$C4
0EAA: 20 A8 FC    122             JSR WAIT
0EAD: AD 00 C0    123             LDA KBD
0EB0: 10 05       124             BPL WARTE1
0EB2: 8D 10 C0    125             STA STROBE
0EB5: 30 03       126             BMI ENDE
                  127 ;
0EB7: CA          128 WARTE1      DEX
0EB8: D0 EE       129             BNE WARTEN
                  130 ;
```

```
                        131  ; Seite löschen und HPRINT als
                        132  ; Ausgabeadresse  eintragen
                        133  ;
0EBA: 20 D8 F3          134  ENDE      JSR HGR2
0EBD: A2 00             135            LDX #$00
0EBF: A0 00             136            LDY #$00
0EC1: 20 FF 0E          137            JSR STRINGPR
0EC4: CA E5 F4          138            ASC "Jetzt koennen Sie ein "
0EDA: F7 E5 EE          139            ASC "wenig schreiben:"
0EEA: 00                140            HEX 00
0EEB: A9 2E             141            LDA #<HPRINT
0EED: 85 36             142            STA CSWL
0EEF: A9 0F             143            LDA #>HPRINT
0EF1: 85 37             144            STA CSWH
0EF3: 4C EA 03          145            JMP CONNECT
                        146  ;
                        147  ; Ende der Demonstration
                        148  ;
                        149  ;
0EF6: A2 28             150  STERN     LDX #40
0EF8: 20 2E 0F          151  STERN1    JSR HPRINT
0EFB: CA                152            DEX
0EFC: 10 FA             153            BPL STERN1
0EFE: 60                154            RTS
                        155  ;
                        156  ; Stringausgabe ähnlich Hertzfeld-Routine
                        157  ;
0EFF: 68                158  STRINGPR  PLA
0F00: 85 04             159            STA PTR
0F02: 68                160            PLA
0F03: 85 05             161            STA PTR+1
0F05: 98                162            TYA
0F06: 48                163            PHA
0F07: 8E FE 0F          164            STX HORZ
0F0A: 8C FF 0F          165            STY VERT
0F0D: A0 01             166            LDY #$01
0F0F: B1 04             167  SPR       LDA (PTR),Y
0F11: F0 06             168            BEQ ENDEN
0F13: 20 2E 0F          169            JSR HPRINT
0F16: C8                170            INY
0F17: D0 F6             171            BNE SPR         ;immer
                        172  ;
0F19: 18                173  ENDEN     CLC
0F1A: 98                174            TYA
0F1B: 65 04             175            ADC PTR
0F1D: 85 04             176            STA PTR
0F1F: A5 05             177            LDA PTR+1
0F21: 69 00             178            ADC #$00
0F23: 85 05             179            STA PTR+1
0F25: 68                180            PLA
0F26: A8                181            TAY
0F27: A5 05             182            LDA PTR+1
0F29: 48                183            PHA
0F2A: A5 04             184            LDA PTR
```

```
0F2C: 48        185             PHA
0F2D: 60        186             RTS
                187  ;
        188  ; Ausgabe von ASCII-Zeichen auf dem Grafikschirm
        189  ; als Bitmuster. Jedes Zeichen liegt in einer
        190  ; 8 x 7 Matrix. Bitposition für jedes Zeichen
        191  ; in einer Tabelle enthalten. Positionierungs-
        192  ; routine rechnet so, daß wie auf dem Textschirm
        193  ; 24 Zeilen a 40 Zeichen geschrieben werden
        194  ; können. Inverse durch Invertierung der Bits.
        195  ; Unterstützt werden ‹Return›, ‹Line-Feed›,
        196  ; ‹Backspace› und ‹Form-Feed› (= HOME).
        197  ; Ein Scroll (Bildschirmrollen) ist nicht
        198  ; implementiert, kann aber ergänzt werden.
                199  ;
0F2E: 08        200  HPRINT  PHP
0F2F: 48        201          PHA
0F30: 8C FD 0F  202          STY YSAVER
0F33: 29 7F     203          AND #$7F
0F35: 8D FC 0F  204          STA ASAVER
0F38: C9 0D     205          CMP #$0D        ;CR
0F3A: F0 1D     206          BEQ CR
0F3C: C9 0A     207          CMP #$0A        ;LF
0F3E: F0 1E     208          BEQ LF
0F40: C9 08     209          CMP #$08        ;BS
0F42: F0 2F     210          BEQ BS
0F44: C9 0C     211          CMP #$0C        ;FF
0F46: F0 49     212          BEQ FORM
0F48: C9 20     213          CMP #$20        ;Leerzeichen
0F4A: 90 21     214          BLT OUT         ;CTRL-Zeichen
0F4C: 20 9E 0F  215          JSR HPRINTER
0F4F: EE FE 0F  216          INC HORZ
0F52: AD FE 0F  217          LDA HORZ
0F55: C9 28     218          CMP #40         ;Dezimal!
0F57: 90 14     219          BLT OUT
0F59: A9 00     220  CR       LDA #0
0F5B: 8D FE 0F  221          STA HORZ
0F5E: EE FF 0F  222  LF       INC VERT
0F61: AD FF 0F  223          LDA VERT
0F64: C9 18     224          CMP #24         ;Dezimal!
0F66: 90 05     225          BLT OUT
0F68: A9 00     226          LDA #0
0F6A: 8D FF 0F  227          STA VERT
0F6D: AC FD 0F  228  OUT      LDY YSAVER
0F70: 68        229          PLA
0F71: 28        230          PLP
0F72: 60        231          RTS
                232  ;
0F73: CE FE 0F  233  BS       DEC HORZ
0F76: 10 0F     234          BPL WEITER
0F78: A9 27     235          LDA #39
0F7A: 8D FE 0F  236          STA HORZ
0F7D: CE FF 0F  237          DEC VERT
0F80: 10 05     238          BPL WEITER
```

```
0F82: A9 17     239             LDA #23
0F84: 8D FF 0F  240             STA VERT
0F87: A9 20     241 WEITER      LDA #$20
0F89: 8D FC 0F  242             STA ASAVER
0F8C: 20 9E 0F  243             JSR HPRINTER
0F8F: B0 DC     244             BGE OUT         ;IMMER
                245 ;
0F91: 20 D8 F3  246 FORM        JSR HGR2
0F94: A9 00     247             LDA #$00
0F96: 8D FF 0F  248             STA VERT
0F99: 8D FE 0F  249             STA HORZ
0F9C: F0 CF     250             BEQ OUT         ;immer
                251 ;
                252 ; Vertikale Position (Zeile) in die
                253 ; entsprechende HGR-Adresse umrechnen
                254 ;
0F9E: AD FF 0F  255 HPRINTER    LDA VERT
0FA1: 4A        256             LSR
0FA2: A8        257             TAY             ;merken
0FA3: 29 03     258             AND #$03
0FA5: 05 E6     259             ORA HPAGE       ;Seite
0FA7: 85 03     260             STA HGRADR+1
0FA9: B9 F0 0F  261             LDA LOTAB,Y     ;Lo aus Tabelle
0FAC: 6A        262             ROR             ;Carry hinein
0FAD: 6D FE 0F  263             ADC HORZ        ;HTAB dazu
0FB0: 85 02     264             STA HGRADR
                265 ;
                266 ; aus dem ASCII-Code des Zeichens seine
                267 ; Position in der Tabelle bestimmen und
                268 ; nach (ADR) speichern.
                269 ; ( POSITION = (ASCII - $20) * 8 )
                270 ;
0FB2: AD FC 0F  271             LDA ASAVER
0FB5: 38        272             SEC
0FB6: E9 20     273             SBC #$20        ;- $20
0FB8: 85 00     274             STA ADR
0FBA: A9 00     275             LDA #0
0FBC: 85 01     276             STA ADR+1
0FBE: 06 00     277             ASL ADR         ;* 8
0FC0: 06 00     278             ASL ADR
0FC2: 26 01     279             ROL ADR+1       ;Übertrag berück-
0FC4: 06 00     280             ASL ADR         ;sichtigen für
0FC6: 26 01     281             ROL ADR+1       ;Hi-Byte
0FC8: 18        282             CLC
0FC9: A9 00     283             LDA #<TABELLE
0FCB: 65 00     284             ADC ADR
0FCD: 85 00     285             STA ADR
0FCF: A9 10     286             LDA #>TABELLE
0FD1: 65 01     287             ADC ADR+1
0FD3: 85 01     288             STA ADR+1
                289 ;
```

```
                    290  ; In einer Schleife die 8 Bytes
                    291  ; des Zeichens ausgeben. Die
                    292  ; nächste Adresse liegt um $400
                    293  ; höher als die alte Adresse.
                    294  ; Da auch Y je Byte um 1 steigt,
                    295  ; wird zur Kompensation nur der
                    296  ; Wert $3FF addiert.
                    297  ;
0FD5: A0 00         298          LDY #0
0FD7: B1 00         299  LOOP   LDA (ADR),Y
0FD9: 49 00         300  INV    EOR #$00       ;$7F = INVERSE
0FDB: 91 02         301          STA (HGRADR),Y
0FDD: C8            302          INY
0FDE: 18            303          CLC
0FDF: A5 02         304          LDA HGRADR
0FE1: 69 FF         305          ADC #$FF
0FE3: 85 02         306          STA HGRADR
0FE5: A5 03         307          LDA HGRADR+1
0FE7: 69 03         308          ADC #$03
0FE9: 85 03         309          STA HGRADR+1
0FEB: C0 08         310          CPY #$08
0FED: 90 E8         311          BLT LOOP
0FEF: 60            312          RTS
                    313  ;
                    314  ;
                    315  ; Tabelle für Lo-Byte der Adresse
                    316  ;
0FF0: 00 00 00      317  LOTAB    HEX 00000000
0FF4: 50 50 50      318          HEX 50505050
0FF8: A0 A0 A0      319          HEX A0A0A0A0
                    320  ;
                    321  ASAVER   DFS 1
                    322  YSAVER   DFS 1
                    323  HORZ     DFS 1
                    324  VERT     DFS 1
                    325  ;
                    326  ; ASCII-ZEICHENSATZ
                    327  ;
1000: 00 00 00      328  TABELLE  HEX 0000000000000000 ;Leerzeichen
1008: 08 08 08      329          HEX 0808080808000800 ; !
1010: 14 14 14      330          HEX 1414140000000000 ; "
1018: 28 94 3F      331          HEX 28943F8A3F850500 ; #
1020: 08 3C 0A      332          HEX 083C0A1C281E0800 ; $
1028: 06 26 10      333          HEX 0626100804323000 ; %
1030: 04 0A 0A      334          HEX 040A0A042A122C00 ; &
1038: 08 08 08      335          HEX 0808080000000000 ; '
1040: 88 84 04      336          HEX 8884040404848800 ; (
1048: 84 88 10      337          HEX 8488101010888400 ; )
1050: 08 2A 1C      338          HEX 082A1C081C2A0800 ; *
1058: 00 08 08      339          HEX 0008083E08080000 ; +
1060: 00 00 00      340          HEX 0000000000080804 ; ,
1068: 00 00 00      341          HEX 0000003E00000000 ; -
1070: 00 00 00      342          HEX 0000000000008C00 ; .
1078: 00 20 10      343          HEX 0020100804020000 ; /
```

```
1080: 1C 92 32  344          HEX 1C92322A26921C00 ; 0
1088: 08 0C 08  345          HEX 080C08080881C00 ; 1
1090: 1C 22 20  346          HEX 1C22201084823E00 ; 2
1098: 3E 90 10  347          HEX 3E90101820221C00 ; 3
10A0: 10 18 14  348          HEX 101814123E101000 ; 4
10A8: 3E 02 1E  349          HEX 3E021E2020221C00 ; 5
10B0: 1C 82 02  350          HEX 1C82021E22221C00 ; 6
10B8: 3E 90 10  351          HEX 3E90108808840400 ; 7
10C0: 1C 22 22  352          HEX 1C22221C22221C00 ; 8
10C8: 1C 22 22  353          HEX 1C22223C20100C00 ; 9
10D0: 00 00 8C  354          HEX 00008C00008C0000 ; :
10D8: 00 00 8C  355          HEX 00008C00008C8884 ; ;
10E0: 10 08 04  356          HEX 1008040204081000 ; ‹
10E8: 00 00 3E  357          HEX 00003E003E000000 ; =
10F0: 04 08 10  358          HEX 0408102010080400 ; ›
10F8: 1C 22 90  359          HEX 1C22908808000800 ; ?
1100: 1C 22 2A  360          HEX 1C222A3A1A023C00 ; §
1108: 08 14 92  361          HEX 081492223E222200 ; A
1110: 1E 22 22  362          HEX 1E22221E22221E00 ; B
1118: 1C 22 02  363          HEX 1C22020202221C00 ; C
1120: 1E 22 22  364          HEX 1E22222222221E00 ; D
1128: 3E 02 02  365          HEX 3E02021E02023E00 ; E
1130: 3E 02 02  366          HEX 3E02021E02020200 ; F
1138: 3C 02 02  367          HEX 3C02020232223C00 ; G
1140: 22 22 22  368          HEX 2222223E22222200 ; H
1148: 1C 08 08  369          HEX 1C08080808081C00 ; I
1150: 20 20 20  370          HEX 2020202020221C00 ; J
1158: 22 12 0A  371          HEX 22120A060A122200 ; K
1160: 02 02 02  372          HEX 0202020202023E00 ; L
1168: 22 36 2A  373          HEX 22362A2A22222200 ; M
1170: 22 22 26  374          HEX 2222262A32222200 ; N
1178: 1C 22 22  375          HEX 1C22222222221C00 ; O
1180: 1E 22 22  376          HEX 1E22221E02020200 ; P
1188: 1C 22 22  377          HEX 1C2222222A122C00 ; Q
1190: 1E 22 22  378          HEX 1E22221E0A122200 ; R
1198: 1C 22 02  379          HEX 1C22021C20221C00 ; S
11A0: 3E 08 08  380          HEX 3E08080808080800 ; T
11A8: 22 22 22  381          HEX 2222222222221C00 ; U
11B0: 22 22 22  382          HEX 2222222222140800 ; V
11B8: 22 22 22  383          HEX 2222222A2A362200 ; W
11C0: 22 92 14  384          HEX 2292140814922200 ; X
11C8: 22 92 14  385          HEX 2292140808080800 ; Y
11D0: 3E 90 10  386          HEX 3E90100804823E00 ; Z
11D8: 3E 06 06  387          HEX 3E06060606063E00 ; Ä
11E0: 00 02 04  388          HEX 0002040810200000 ; Ö
11E8: 3E 30 30  389          HEX 3E30303030303E00 ; Ü
11F0: 00 00 08  390          HEX 0000081422000000 ; ↑
11F8: 00 00 00  391          HEX 000000000000007F ; _
1200: 04 08 10  392          HEX 0408100000000000 ; "
1208: 00 00 1C  393          HEX 00001C203C223C00 ; a
1210: 02 02 1E  394          HEX 02021E2222221E00 ; b
1218: 00 00 3C  395          HEX 00003C0202023C00 ; c
1220: 20 20 3C  396          HEX 20203C2222223C00 ; d
1228: 00 00 1C  397          HEX 00001C223E023C00 ; e
```

```
1230: 18 24 04  398          HEX 1824041E04040400 ; f
1238: 00 00 1C  399          HEX 00001C22223C201C ; g
1240: 02 02 1E  400          HEX 02021E2222222200 ; h
1248: 08 00 0C  401          HEX 08000C0808081C00 ; i
1250: 10 00 18  402          HEX 100018101010120C ; j
1258: 02 02 22  403          HEX 020222120A162200 ; k
1260: 0C 08 08  404          HEX 0C08080808081C00 ; l
1268: 00 00 36  405          HEX 0000362A2A2A2200 ; m
1270: 00 00 1E  406          HEX 00001E2222222200 ; n
1278: 00 00 1C  407          HEX 00001C2222221C00 ; o
1280: 00 00 1E  408          HEX 00001E22221E0202 ; p
1288: 00 00 3C  409          HEX 00003C22223C2020 ; q
1290: 00 00 3A  410          HEX 00003A0602020200 ; r
1298: 00 00 3C  411          HEX 00003C021C201E00 ; s
12A0: 04 04 1E  412          HEX 04041E0404241800 ; t
12A8: 00 00 22  413          HEX 0000222222322C00 ; u
12B0: 00 00 22  414          HEX 00002292148C0800 ; v
12B8: 00 00 22  415          HEX 000022222A2A3600 ; w
12C0: 00 00 22  416          HEX 0000221408142200 ; x
12C8: 00 00 22  417          HEX 00002292148C0886 ; y
12D0: 00 00 3E  418          HEX 00003E1008043E00 ; z
12D8: 38 0C 0C  419          HEX 380C0C060C0C3800 ; ä
12E0: 08 08 08  420          HEX 0808080808080808 ; ö
12E8: 0E 18 18  421          HEX 0E18183018180E00 ; ü
12F0: 00 2C 1A  422          HEX 002C1A0000000000 ; ß
12F8: 7F 7F 7F  423          HEX 7F7F7F7F7F7F7F7F ; Block
```

Das Herz des Programms ist die HPRINT-Routine. Sie können sie auch ohne die voranstehenden Teile als selbständiges Programm benutzen. Der zugehörige Zeichensatz darf an beliebiger Stelle im RAM liegen, wenn seine Startadresse in Zeile 283 (Lo-Byte) und 286 (Hi-Byte) eingetragen wird.

Damit Sie HPRINT als Unterroutine ähnlich wie COUT aufrufen können, müssen die innerhalb des Programms benutzten Register (Akku und Y-Reg.) sowie der Prozessorstatus gerettet (Zeile 200-202) und später zurückgeholt werden (Z. 228-230). Das auszugebende Zeichen wird wie bei COUT im Akkumulator übergeben. Zunächst wird das Bit7 ausgeblendet und der ASCII-Code für die spätere Ausgabe der Bit-Grafik zwischengespeichert.

Einige Zeichen, wie z.B. der Rückwärtspfeil, verlangen eine Sonderbehandlung. Auf sie wird durch einfache Vergleiche getestet und bei Vorhandensein in eine Spezialroutine verzweigt. Bei einem <RETURN> wird die horizontale Cursorposition (HORZ) auf Null gesetzt und die vertikale Position (VERT) um 1 erhöht. Wird dabei die unterste Bildschirmzeile überschritten, wird der Cursor in die erste Zeile des Bildschirms gesetzt. Ein rollen (scroll) wie beim Textbildschirm findet nicht statt. Wenn sie dieses erreichen wollen, müssen Sie nach dem Vergleich (CMP #24) bei Bedarf zu einer Scroll-Routine verzweigen. Es eignet sich dazu z.B. „HIRES-SCROLL“ aus dem Fensterprogramm.

Bei einem Zeilenvorschub (Line-Feed LF) wird nur der vertikale Cursorwert erhöht, der horizontale bleibt erhalten. Ein Rückwärtspfeil (Backspace BS) verringert HORZ um 1. Wird dabei der linke Rand überschritten (HORZ = $FF), dann wird der Cursor in die letzte Position der darüberliegenden Zeile gesetzt. Waren wir schon in der obersten Zeile, springt der Cursor an das Ende der untersten Zeile. Der Bildschirm ist also wie eine Rolle aufgebaut.
Ein Seitenvorschub (Form-Feed FF) löscht den Bildschirm und setzt den Cursor an den Anfang der ersten Zeile wie bei einem HOME des Textbildschirms.
Alle übrigen Controll-Zeichen werden nicht behandelt. Durch den Vergleich mit dem Leerzeichen ($20), dem druckbaren Zeichen mit dem kleinsten ASCII-Wert, werden sie ausgesondert (BLT OUT).

Die Ausgabe des zum Zeichen gehörenden Bitmusters besorgt die Routine HPRINTER. Jedes Zeichen beansprucht die acht zu einer Gruppe gehörenden Bildschirmzeilen. Deshalb wird zunächst die Adresse der Grundlinie (oben!) berechnet. Dazu dient derselbe Algorithmus wie in HIRES-SCROLL vom Fenster-Programm. Das Drucken eines Zeichens beginnt also immer von oben. Deshalb sind auch die Bytes in der Zeichensatz-Tabelle so angeordnet, daß das oberste Byte an erster Stelle steht.
Der ASCII-Wert bestimmt die Position eines Zeichens in der Tabelle. Da das erste Zeichen der Tabelle, das Leerzeichen, bereits den ASCII-Wert $20 besitzt, ergibt sich die Formel zur Positionsberechnung zu
POSITION = (ASCII-Wert - $20) * 8.
Die Multiplikation mit 8 rührt von der Tatsache her, daß jedes Zeichen 8 Bytes lang ist. Der ASCII-Wert wird aus seinem Zwischenspeicher zurückgeholt (Z. 271), um dann um $20 vermindert zu werden. Die Multiplikation mit 8 geschieht durch dreimaliges Linksschieben. Nach der ersten Multiplikation mit 2 muß das Ergebnis < 192 sein. Das nächste ASL kann bereits das Fassungsvermögen eines Bytes überschreiten, da ein Ergebnis < 384 dabei herauskommen kann. Deshalb muß ein eventuell sich im Carry-Bit gebildeter Übertrag mit ROL ADRESSE+1 in das Hi-Byte gebracht werden. Auch das nächste Linksschieben muß jetzt beide Bytes berücksichtigen, da ein Ergebnis < 768 sich ergeben kann.
Um zu der absoluten Adresse zu gelangen, wird jetzt noch die Startadresse der Tabelle addiert.
Die Ausgabe der 8 Bytes geschieht in einer Schleife, in der das Y-Register als Laufvariable dient. Das Byte wird von der berechneten Tabellenposition geladen (Z. 299) und auf die vorher berechnte Grundlinie gesetzt (Z. 301). Das Y-Register wird anschließend um 1 erhöht, damit LDA (ADR),Y auf das nächste Byte der Tabelle zeigt. Die Adresse der nächsten Zeile liegt um jeweils $400 über der vorangegangenen. Deshalb muß $400 zu (HGRADR) addiert

werden. Da die Abspeicherung aber mit der indirekten indizierten Adressierung durch das Y-Register erfolgt, ist eine kleine Korrektur erforderlich: da Y um 1 größer geworden ist, wird $400 – $1 = $3FF zur alten Adresse dazugezählt.

Die Zeile 300 muß ich Ihnen noch erklären. So, wie sie jetzt im Programm steht, bewirkt sie nichts. Sie dient lediglich als Platzhalter. Das aufrufende Programm kann allerdings den Operanden von $00 nach $7F verwandeln. Das entstehende EOR #$7F invertiert das Byte im Akku (mit Ausnahme des Farbbits), bevor es auf den Bildschirm ausgegeben wird. Dies bewirkt eine inverse Darstellung. Durch das passende setzen dieses Bytes können sie also zwischen den Attributen Normal und Inverse wechseln. Blinkende Zeichen sind nur durch ständiges Löschen und Neuzeichnen zu verwirklichen, eine auch in Assembler mühsame und aufwendige Methode. Flash ist deshalb in diesem Programm nicht vorgesehen.

3.10 Der Kopfstand der Bytes

Wenn Sie sich viel Mühe gemacht und einen eigenen Zeichensatz entworfen haben, werden Sie sich sicher darüber freuen, daß Sie ihn nicht nur für Ihre HIRES-Grafiken benutzen können. Es gibt die Möglichkeit, diesen Zeichensatz auch auf einen Matrixdrucker zu bringen, ohne daß dieser ein spezielles ladbares Zeichensatz-RAM besitzen muß. Einzige Voraussetzung ist die Fähigkeit, eine Grafikzeile mit 640 Punkten drucken zu können.
Da sich die verschiedenen Druckerfabrikate in ihrer Ansteuerung unterscheiden, mußte das folgende Programm für einen speziellen Drucker geschrieben werden. Es läuft unverändert auf einem EPSON FX-80 und allen befehlskompatiblen Geräten. Als Interface dient ein EPSON Parallel-Interface in Slot 1. Es muß nicht die Grafik-Version sein. Das NPara2 von Brosius & Köhler wurde ebenfalls erfolgreich getestet.
Das Programm „HGR-Drucker" liegt ab $8F00 im RAM des Apple. Darüber ist nach $9300 der Zeichensatz zu laden. Dies kann der Zeichensatz aus Kapitel 3.8 sein, ein selbst entworfener oder einer von der „DOS Programmers Toolkit"-Diskette. Hier ist besonders „COLOSSAL.SET" sehr schön.
Vor dem eigentlichen Programm liegt ein kurzes Startprogramm, daß nur einmal zur Initialisierung benötigt wird und dann überschrieben werden kann. Wegen der Art des Aufrufs ist es nur unter DOS 3.3 und nicht unter ProDOS

lauffähig. Wie Sie das Programm auch unter ProDOS benutzen können, wird später in diesem Band erklärt.
Betrachten wir zunächst die Zeilen 28 bis 38. Es wird der Binärmodus hergestellt und dann die Startadresse des eigentlichen Programms ($8F00) sowohl nach LINNUM als auch nach CSWL/H geschrieben. Das folgende JSR HIMEM1 holt diese Adresse wieder aus LINNUM und trägt sie sowohl als HIMEM-Wert für BASIC als auch als TOP OF STRING-Pointer ein. Mehr über diesen letzten Wert erfahren Sie im Kapitel 5. Dadurch wird das Maschinenprogramm vor dem Überschreiben durch die Stringvariablen eines BASIC-Programms geschützt. Zugleich bedeutet es aber auch, daß alle Stringvariablen gelöscht werden. Sie sollten deshalb „HGR-Drucker" aus einem BASIC-Programm immer gleich zu Beginn aufrufen, bevor irgendwelche Variablen definiert wurden.

HIMEM1 $F28C
Setzt neuen HIMEM-Wert, wenn dieser kleiner als der alte ist. Andernfalls erfolgt eine Fehlermeldung.

Eingabe: LINNUM ($0050/51) = neuer HIMEM-Wert

Ausgabe: HIMEM ($0073/74) und FRETOP ($006F/70) gesetzt

CSW steht für Character Output Switch und bezeichnet einen Zeiger auf der Nullseite ($0036/37). Daneben gibt es noch KSW, den Keyboard Switch, zu dem die Adresse $0038/39 gehören. Fast alle Ein- und Ausgaben des Monitors und des BASIC-Interpreters gehen über diese beiden Vektoren. Der erste Befehl von COUT lautet z.B. „JMP (CSWL)". In CSWL/H steht dann die eigentliche Ausgabeadresse. Durch ein Umsetzen dieser Zeiger hat es Steve Wozniak bei der Konstruktion des Apple möglich gemacht, die Ein- und Ausgaben umzulenken. Nur dadurch war es z.B. möglich, DOS (und ProDOS) in den normalen Datenverkehr mit einzubeziehen. Ist DOS aktiv, so zeigt CSW nach $9EBD und KSW nach $9E81. Unter ProDOS lauten die Werte $B84B bzw. $B84E.
Das DOS überprüft, ob es sich bei einer Ein-/Ausgabe um einen DOS-Befehl handelt. Erkennungszeichen bei der Ausgabe ist ein Ctrl-D (CHR$(4)) vor einem DOS-Befehl, der daraufhin innerhalb des DOS verarbeitet wird. Handelt es sich um keinen DOS-Befehl, wird er an die entsprechende Routine im Monitor oder auf einer aktiven Interface-Karte weitergeleitet. Bis zum Ende der Befehlsabarbeitung klinkt sich DOS dabei aus dem Datenverkehr aus, um dann erneut den Vektor in sich selbst hinein zu setzen.

Woher weiß nun DOS, wem es das Kommando zu übergeben hat?
Wenn DOS aktiviert wird (z.B. nach PR #6), rettet es die alten Vektoren und schreibt sie in seine eigenen Ein/Ausgabevektoren bei $AA53 bis $AA56 (DOS 3.3) bzw. $BE30 bis $BE33 (ProDOS).
Wenn wir die Ausgabe wie in unserem Programm nach START von HGR-Drukker umleiten wollen, damit die Ausgabe unseres Zeichensatzes auf den Drucker geht, müssen wir die Adresse von START nach CSWL/H schreiben und anschließend DOS von der Änderung informieren, da das DOS sonst hartnäckig immer wieder die alte Adresse zurückträgt. Mit JSR CONNECT über einen Vektor auf Seite 3 ($03EA) geschieht dies bei DOS 3.3, das daraufhin die Adresse von START in seine eigenen Ausgabevektoren übernimmt. Unter ProDOS geht das ganze etwas anders. Hier muß die Adresse von START nach $BE30/31 kopiert werden. (Die Eingabeadresse von ProDOS ist in $BE32/33 abgelegt)
Unser Vorspann-Programm ruft schließlich noch die Routine CLEAR auf, die im Hauptprogramm steht und den Druckpuffer löscht.

Nach so langer Vorrede soll jetzt erst einmal HGR-Drucker folgen.

HGR-DRUCKER

```
                    1   ;*******************************
                    2   ;  HGR-Drucker für EPSON FX-80 *
                    3   ;     Dr. Jürgen B.  Kehrel    *
                    4   ;    Heidelberg, Januar 1986   *
                    5   ;*******************************
                    6   ;
                    7   ; Version für EPSON Parallel-
                    8   ; Interface oder Brosius&Köhler
                    9   ; NPara2 in Slot 1 des Apple
                   10   ;
                   11   ;
                   12   ADRESSE   EQU $0006       ;+$0007
                   13   CSWL      EQU $0036
                   14   ACC       EQU $0045
                   15   LINNUM    EQU $0050
                   16   CONNECT   EQU $03EA
                   17   HIMEM1    EQU $F28C
                   18   COUT1     EQU $FDF0
                   19   RESTORE   EQU $FF3F
                   20   SAVE      EQU $FF4A
                   21   ;
                   22   TABELLE   EQU $9300       ;-$95FF
                   23   READY     EQU $C1C1       ;Slot 1
                   24   STROBE    EQU $C090       ;Parallelinterf.
                   25   ;
                   26             ORG $8F00-$17
                   27   ;
8EE9: D8           28             CLD
8EEA: A9 00        29             LDA #<START
```

```
8EEC: 85 50      30              STA LINNUM
8EEE: 85 36      31              STA CSWL
8EF0: A9 8F      32              LDA #>START
8EF2: 85 51      33              STA LINNUM+1
8EF4: 85 37      34              STA CSWL+1
8EF6: 20 8C F2   35              JSR HIMEM1
8EF9: 20 EA 03   36              JSR CONNECT
8EFC: 20 AF 8F   37              JSR CLEAR
8EFF: 60         38              RTS
                 39   ;
8F00: 20 4A FF   40   START      JSR SAVE        ;Register retten
8F03: A5 45      41              LDA ACC         ;Akku zurückladen
8F05: 20 F0 FD   42              JSR COUT1       ;40Z Bildschirm
8F08: 29 7F      43              AND #$7F        ;Bit 7 löschen
8F0A: C9 20      44              CMP #$20        ;Leerzeichen
8F0C: B0 10      45              BGE DRUCKBAR    ;für Druckzeile
8F0E: C9 0D      46              CMP #$0D        ;Return?
8F10: D0 08      47              BNE FERTIG      ;ausgeben
8F12: 20 78 8F   48              JSR PRINT       ;Grafikzeile
8F15: A9 0A      49              LDA #$0A        ;Zeilenvorschub
8F17: 20 D2 8F   50              JSR OUTPUT      ;ausgeben
8F1A: 20 3F FF   51   FERTIG     JSR RESTORE     ;Register zurück
8F1D: 60         52              RTS
                 53   ;
                 54   ;
                 55   ; Position in der Tabelle bestimmen
                 56   ; Leerzeichen ist 1. Zeichen
                 57   ;
8F1E: 38         58   DRUCKBAR   SEC             ;Druckzeichen im Akku
8F1F: E9 20      59              SBC #$20        ;ASCII für Leerzeichen
8F21: 85 06      60              STA ADRESSE     ;merken
8F23: A9 00      61              LDA #$00        ;Hi-Byte auf
8F25: 85 07      62              STA ADRESSE+1   ;Null setzen
8F27: 06 06      63              ASL ADRESSE     ;*2 (<192)
8F29: 06 06      64              ASL ADRESSE     ;*4 (<384)
8F2B: 26 07      65              ROL ADRESSE+1   ;ev. Übertrag
8F2D: 06 06      66              ASL ADRESSE     ;*8 (<768)
8F2F: 26 07      67              ROL ADRESSE+1   ;ev. Übertrag
                 68   ;
                 69   ; Zum berechneten Grundwert in der Tabelle
                 70   ; die Anfangsadresse der Tabelle addieren
                 71   ; für die tatsächliche Speicherposition
                 72   ;
8F31: 18         73              CLC
8F32: A9 00      74              LDA #<TABELLE   ;Lo-Byte
8F34: 65 06      75              ADC ADRESSE
8F36: 85 06      76              STA ADRESSE
8F38: A9 93      77              LDA #>TABELLE   ;Hi-Byte
8F3A: 65 07      78              ADC ADRESSE+1
8F3C: 85 07      79              STA ADRESSE+1
                 80   ;
                 81   ; Die 8 Bytes des Zeichens in einen
                 82   ; Zwischenspeicher laden für die
                 83   ; dann folgende Umcodierung
                 84   ;
```

```
8F3E: A0 07      85              LDY #$07        ;8 Bytes
8F40: B1 06      86   LADEN     LDA (ADRESSE),Y
8F42: 29 7F      87              AND #$7F        ;Bit 7 löschen
8F44: 99 DB 8F   88              STA ZEICHEN,Y   ;Speicher
8F47: 88         89              DEY
8F48: 10 F6      90              BPL LADEN       ;weiter
                 91   ;
                 92   ; Die Bitwerte von horizontaler
                 93   ; Anordnung (Bildschirm) umrech-
                 94   ; nen in vertikale Anordnung
                 95   ; für den Matrixdrucker
                 96   ;
8F4A: A0 07      97              LDY #$07        ;8 Bit im Byte
8F4C: 18         98   LOOP      CLC             ;Carry löschen
8F4D: A9 00      99              LDA #$00        ;Akku initialisieren
8F4F: A2 07      100             LDX #$07        ;8 Bytes pro Zeichen
8F51: 7E DB 8F   101  ROLLEN    ROR ZEICHEN,X   ;entspr. Bit in
8F54: 6A         102             ROR             ;den Akku rollen
8F55: CA         103             DEX             ;alle Bytes?
8F56: 10 F9      104             BPL ROLLEN      ;Nein, weiter
8F58: 20 60 8F   105             JSR ABLAGE      ;im Puffer ablegen
8F5B: 88         106             DEY             ;alle Bits?
8F5C: 10 EE      107             BPL LOOP        ;Nein, weiter
8F5E: 30 BA      108             BMI FERTIG      ;JA
                 109  ;
                 110  ; Umcodierte Bytes in einem Zwischenpuffer
                 111  ; merken, bis entweder ein Return folgt
                 112  ; oder der Puffer voll ist (640 Bytes)
                 113  ;
8F60: 8D FF FF   114  ABLAGE    STA $FFFF       ;Dummy-Adresse
8F63: EE 61 8F   115             INC ABLAGE+1    ;Lo-Byte weitersetzen
8F66: D0 03      116             BNE TEST        ;Übertrag?
8F68: EE 62 8F   117             INC ABLAGE+2    ;Hi-Byte ebenso
8F6B: AD 61 8F   118  TEST      LDA ABLAGE+1    ;Puffer-Voll testen
8F6E: C9 68      119             CMP #<PUFFER+$280 ;Pufferende Lo
8F70: AD 62 8F   120             LDA ABLAGE+2
8F73: E9 92      121             SBC #>PUFFER+$280 ;Pufferende Hi
8F75: B0 01      122             BGE PRINT       ;>=, also ausgeben
8F77: 60         123             RTS
                 124  ;
                 125  ; Zeile im Zwischenpuffer ausgeben
                 126  ;
                 127  ; Zahl der Zeichen berechen, da
                 128  ; der Drucker diese wissen muß
                 129  ;
8F78: 38         130  PRINT     SEC
8F79: AD 61 8F   131             LDA ABLAGE+1    ;Lo-Byte
8F7C: E9 E8      132             SBC #<PUFFER    ;Start abziehen
8F7E: 8D E6 8F   133             STA AUSGABE+3   ;merken
8F81: AD 62 8F   134             LDA ABLAGE+2    ;Hi-Byte
8F84: E9 8F      135             SBC #>PUFFER
8F86: 8D E7 8F   136             STA AUSGABE+4   ;Länge in Byte
                 137  ;
                 138  ; Druckroutine auf den Startwert setzen
                 139  ;
```

```
8F89: A9 E3     140          LDA #<AUSGABE
8F8B: 8D 94 8F  141          STA DRUCKEN+1  ;Lo
8F8E: A9 8F     142          LDA #>AUSGABE
8F90: 8D 95 8F  143          STA DRUCKEN+2  ;Hi
                144 ;
                145 ; Die Zeichen des Vorspanns und alle
                146 ; Zeichen des Zwischenpuffers senden
                147 ;
8F93: AD FF FF  148 DRUCKEN  LDA $FFFF      ;Dummy-Adresse
8F96: 20 D2 8F  149          JSR OUTPUT     ;ausgeben
8F99: EE 94 8F  150          INC DRUCKEN+1  ;Lo weitersetzen
8F9C: D0 03     151          BNE TEST1      ;Übertrag?
8F9E: EE 95 8F  152          INC DRUCKEN+2  ;Ja, auch Hi
8FA1: AD 94 8F  153 TEST1    LDA DRUCKEN+1  ;Zwischenpufferende?
8FA4: CD 61 8F  154          CMP ABLAGE+1   ;Lo
8FA7: AD 95 8F  155          LDA DRUCKEN+2
8FAA: ED 62 8F  156          SBC ABLAGE+2   ;Hi
8FAD: 90 E4     157          BLT DRUCKEN    ;noch kleiner
                158 ;
                159 ; Zwichenpuffer löschen und Zeiger
                160 ; (Dummy-Adressen) auf Startwert setzen
                161 ;
8FAF: A9 68     162 CLEAR    LDA #<PUFFER-$80
8FB1: 85 06     163          STA ADRESSE
8FB3: A9 8F     164          LDA #>PUFFER-$80
8FB5: 85 07     165          STA ADRESSE+1  ;auf Startwert
8FB7: A2 03     166          LDX #$03       ;Schleifenzähler
8FB9: A0 80     167          LDY #$80       ;Startwert
8FBB: A9 00     168          LDA #$00       ;Null zum Löschen
8FBD: 91 06     169 CLOOP    STA (ADRESSE),Y
8FBF: C8        170          INY
8FC0: D0 FB     171          BNE CLOOP
8FC2: E6 07     172          INC ADRESSE+1  ;Hi-Byte weiter
8FC4: CA        173          DEX            ;Rundenzähler
8FC5: D0 F6     174          BNE CLOOP      ;Noch nicht fertig
                175 ;
8FC7: A9 E8     176          LDA #<PUFFER
8FC9: 8D 61 8F  177          STA ABLAGE+1   ;initialisieren
8FCC: A9 8F     178          LDA #>PUFFER
8FCE: 8D 62 8F  179          STA ABLAGE+2
8FD1: 60        180          RTS
                181 ;
                182 ; Zeichen im Akku an den Drucker
                183 ; ausgeben. Routine ist Slot-
                184 ; und Interfaceabhängig !!!
                185 ; Gilt für EPSON-Interface in Slot 1
                186 ;
8FD2: 2C C1 C1  187 OUTPUT   BIT READY      ;Ready?
8FD5: 30 FB     188          BMI OUTPUT     ;Nein
8FD7: 8D 90 C0  189          STA STROBE     ;ausgeben + Strobe
8FDA: 60        190          RTS            ;fertig
                191 ;
                192 ; Zwischenpeicher für 8 Bytes eines Zeichens
                193 ;
```

```
                  194  ZEICHEN   DFS 8
                  195  ;
                  196  ; Text, der dem Drucker vor jeder Zeile ge-
                  197  ; sendet werden muß. Beim EPSON „ESCAPE, *,
                  198  ; 4, Zeichenzahl in 2 Bytes" für den
                  199  ; 8-Nadel Bitmuster-Modus
                  200  ;
8FE3: 1B 2A 04    201  AUSGABE   HEX 1B2A04FFFF ;ESC * 4 Dummy
8FE8: 00          202  PUFFER    HEX 00         ;Pufferstart
```

Damit Sie dieses Programm auch gleich ausprobieren können, folgt ein kurzes DEMO-Programm. Dazu muß auf der selben Diskette der Zeichensatz CHAR.SET.OBJ vorhanden sein, den Sie, wenn Sie nicht die Begleitdiskette zu Band 2 gekauft haben, aus dem Programm HIRES-Schriften-Generator aus Kapitel 3.8 herausnehmen können.

```
10  PRINT  CHR$ (4)"BLOAD CHAR.SET.OBJ,A$9300"
20  PRINT  CHR$ (4)"BRUN HGR-DRUCKER.OBJ"
30  FOR I = 32 TO 127: PRINT  CHR$ (I);: NEXT
35  PRINT
40  PRINT  CHR$ (4)"CATALOG"
50  END
```

Ändern Sie Zeile 10, wenn Sie einen anderen Zeichensatz benutzen wollen. Um den Ausgabevektor wieder auf seinen normalen Wert zu setzen, drücken Sie bitte <RESET> oder <CTRL-RESET>. Falls Sie einen etwas sanfteren Weg, z.B. aus einem Programm heraus, vorziehen: tragen Sie $FDF0 bei CSW ein und rufen Sie CONNECT auf. Dann ist der 40-Zeichenschirm wieder angekoppelt. War vorher die 80-Zeichen-Karte aktiv, so lautet die Adresse $C307.

Schauen wir uns wieder gemeinsam einige Programmteile an. Das Hauptprogramm liegt in Zeile 40 bis 52. Von dort aus werden bei Bedarf andere Programmteile aufgerufen oder angesprungen. Die Monitor-Routine SAVE rettet alle Register, den Stackpointer und den Prozessorstatus auf die Zero-Page ab $0045. RESTORE wird sie später wieder zurückholen mit Ausnahme des Stackpointers, der nicht restauriert wird. Die dazwischenliegenden Zeilen sollten Sie an den HIRES-Schriften-Generator erinnern: Bit 7 wird gelöscht und dann mit dem Leerzeichen verglichen, um Controll-Zeichen auszusondern. Alle Zeichen ab dem Leerzeichen werden in der Routine DRUCKBAR weiter behandelt. Von den Controll-Zeichen hat nur noch das <RETURN> eine besondere Bedeutung. Es führt zum sofortigen Ausdruck der Druckzeile und einem anschließenden Zeilenvorschub ($0A = LF) auf dem Drucker.

SAVE $FF4A
Rettet die Prozessor-Register

Eingabe: –

Ausgabe:
$0045 = Akku, $0046 = X-Reg, $0047 = Y-Reg
$0048 = Prozessorstatus, $0049 = Stackpointer
Setzt Binärmodus (CLD)

RESTORE $FF3F
Stellt die Prozessor-Register wieder her

Eingabe: –

Ausgabe:
Akku = $0045, X-Reg = $0046, Y-Reg = $0047
Prozessorstatus = $0048
Der Stackpointer bleibt unverändert!

Auch DRUCKBAR sollte bei Ihnen Erinnerungen wachrufen. Die Zeilen 58 bis 79 sind praktisch identisch mit den Zeilen 272 bis 288 des Schriften-Generators. Nun kommt aber Neues hinzu. Die 8 Bytes, die zu dem Zeichen gehören, werden hintereinander in einen Zwischenspeicher (ZEICHEN) geladen (Z. 85–90). Dabei wird mit AND $7F das Farbbit gelöscht, da es für die Druckerausgabe nicht gebraucht wird.
In den Zeilen 97 bis 107 wird die horizontale Anordnung der Bits, die für die Ausgabe auf den Bildschirm erforderlich ist, in eine vertikale für den Drucker umgerechnet. Auf dem Bildschirm werden die Bytes nacheinander von oben nach unten ausgegeben. Der Kopf des Druckers bewegt sich dagegen vertikal über eine Zeile. Bei einer Bewegung von links nach rechts muß zunächst das Bit 0 von allen 8 Datenbytes zusammengefaßt ausgegeben werden. Wir fangen mit Bit 0 an, da wegen der HIRES-Darstellung alle Zeichen spiegelverkehrt codiert wurden. Als nächstes folgen alle Bit 1 usw. bis Bit 7. Die zwei Schleifen LOOP (äußere Schleife) und ROLLEN (innere Schleife) besorgen diese Umstellung.

Da beim EPSON das Bit 7 oben gedruckt wird, müssen die Bits so in den Akkumulator gerollt (ROR) werden, daß als letztes das Bit aus dem ersten Byte hineingebracht wird. Wir starten deshalb in der inneren Schleife immer mit X = 7, um mit dem letzten Byte anzufangen. ROR ZEICHEN,X rollt das gewünschte Bit ins Carry-Bit, von wo es mit ROR in den Akkumulator geschoben und dort gesammelt wird. Sind alle 8 Byte durch, wird der Akku, der jetzt ein gültiges Druckerbyte enthält, in einen Druckzwischenspeicher gepackt. Die äußere Schleife mit dem Y-Register als Laufvariablen sorgt dafür, daß alle 8 Bits behandelt werden. Probieren Sie diese Routine ruhig mit ein paar Werten im IDUS aus und lassen sich dabei 6 Bytes von ZEICHEN in der freien Anzeige unten rechts anzeigen.

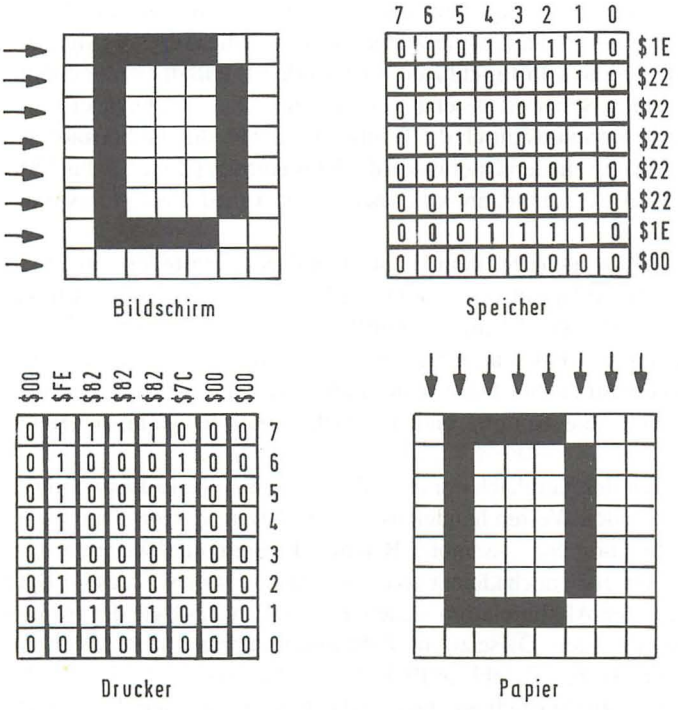


Abb. 7: Bitmusterwandlung

ABLAGE speichert das im Akkumulator übergebene Zeichen im Druckpuffer. Hier werden alle Bytes gesammelt, bis eine Druckzeile voll ist oder bis ein <RETURN> erfolgt. In eine Zeile passen 80 Zeichen, die je 8 Bytes beanspruchen, also insgesamt 640 Bytes. Genauso groß ist der Druckspeicher.
Die erste Besonderheit dieser Routine ist die Zeile 114, die sogenannten selbstmodifizierenden Code enthält. Das ist ein Code, der sich während der Laufzeit eines Programms ändert, weil er vom Programm selbst modifiziert wird. Das wäre so, als wenn Sie in einem BASIC-Programm durch das Programm selbst eine Adresse für einen POKE-Befehl errechnen lassen würden, um sie dann in das laufende Programm schreiben und danach ausführen zu lassen. Die Adresse für den STA-Befehl wird vom Programm während des Laufs errechnet. Der Wert $FFFF steht nur dort, damit der Assembler beim assemblieren für STA die absolute Adressierung berechnet und dann mit den FF FF zwei Bytes im Code freihält, in die später die eigentliche Adresse eingesetzt wird. Wichtig bei jedem selbstmodifizierenden Code ist, daß der Code initialisiert, d.h. auf seinen Startwert gesetzt werden muß. Dies geschieht erstmals beim Aufruf des Programms durch die Routine CLEAR, die die Startadresse des Puffers (hier $8FE8) in den Operanden des STA einträgt (Zeile 176 bis 179). Die Routine ABLAGE setzt dann selber bei jedem Aufruf diese Adresse um eins weiter.
Irgendwann ist der Druckpuffer einmal voll. Um dies festzustellen, findet ein Test statt. Wenn die Ablageadresse erhöht worden ist, wird sie mit der Adresse des Pufferendes verglichen. Ist diese erreicht, wird die Druckroutine PRINT aufgerufen, die alle 640 Bytes ausgibt, anschließend den Puffer wieder löscht und in Zeile 114 erneut die Startadresse des Puffers einträgt. Ist das Ende noch nicht erreicht, endet die Routine und das nächste Zeichen kann behandelt werden.
Interessant ist noch der Vergleich der augenblicklichen Ablageadresse mit der Endadresse. Bei beiden Werten handelt es sich um 2-Byte Zahlen. Durch eine geschickte Kombination von Assembler-Befehlen kann in nur 4 Zeilen festgestellt werden, ob die erste noch kleiner als die zweite ist. Dazu wird das Lo-Byte der ersten Zahl in den Akkumulator geladen, um mit dem Lo-Byte der zweiten Zahl verglichen zu werden. Diese zweite Zahl wird im übrigen vom Assembler errechnet, der durch den Befehl „<PUFFER+$280“ zuerst die Länge des Puffers ($280 = 640) zur Startadresse zählt und von der Summe nur das Lo-Byte nimmt. War das Lo-Byte von ABLAGE kleiner als das von „ENDE“, ist danach das Carry-Bit gelöscht, ansonsten gesetzt. Jetzt wird das Hi-Byte von ABLAGE geladen und das Hi-Byte von „ENDE“ subtrahiert. Ist vorher das Lo-Byte kleiner gewesen, so ist das Carry-Bit gelöscht, sodaß zusätzlich eine 1 subtrahiert wird. Im anderen Fall wird kein Übertrag berücksichtigt. Ist am Ende

dieser Zeilen das Carry-Bit gesetzt, ist die erste Zahl gleich groß oder größer als die zweite. Ist das Carry-Bit gelöscht, war die erste Zahl kleiner.
Nehmen wir drei Beispiele: die erste Zahl sei nacheinander $6020, $6030 und $6040. Die zweite Zahl sei immer $6030.

```
Akku:            $20          $30          $40
Vergleich mit    $30          $30          $30
Carry danach:      0            1            1

Akku:            $60          $60          $60
Subtraktion von  $60          $60          $60
Übertrag:          1            0            0
Ergebnis:        $FF          $00          $00
Carry danach       0            1            1
```

Denken Sie daran, daß der Übertrag bei einer Subtraktion gleich dem invertierten Carry-Bit ist (Borgeflagge!).

Die PRINT-Routine berechnet durch Subtraktion des Pufferstarts von der letzten Ablageposition die Anzahl der Zeichen im Puffer, da diese dem Drucker vor dem Ausdruck der Zeile übermittelt werden muß. DRUCKEN enthält selbstmodifizierenden Code in genau der selben Art wie ABLAGE. Hier wird die Initialisierung von den Zeilen 140 bis 143 vorgenommen. Es werden solange Zeichen aus dem Druckpuffer an den Drucker geschickt, bis alle Bytes gesendet wurden (Vergleich mit ABLAGE). Dann wird der Puffer gelöscht (wir besprechen diesen Teil noch) und der selbstmodifizierende Code von ABLAGE wieder auf den Anfangswert gesetzt.
Die Druckroutine OUTPUT ist Interface-spezifisch und gilt in dieser Form nur für die oben genannten Karten. Das Byte wird dabei direkt auf die Hardware-Adressen der Interface-Karte geschrieben. Für andere Karten müssen Sie dort deren Übergabe einsetzen, die Sie in der Betriebsanleitung des Interfaces finden. Unter den folgenden Routinen müßte auch eine für Sie dabei sein, z.B.:

```
OUTPUT  PHA              ;Akku retten
        LDY #$10         ; Slot 1
OUTLOOP LDA $C080,Y      ;Status laden
        AND #%0000 0111  ;NO PAPER, SELECT und
        CMP #%000000010  ;ACKNOWLEDGE testen
        BNE OUTLOOP      ;Drucker nicht bereit
        PLA
        STA $C080,Y      ;Zeichen ausgeben
        RTS              ;fertig
```

oder

```
OUTPUT  BIT $C091       ; Slot 1
        BMI OUTPUT
        STA $C091       ;z.B. C.ITOH 8510A
        STA $C092
        STA $C094
        RTS
```

oder

```
OUTPUT  PHA             ; retten
OUTLOOP LDA $C091       ; Slot 1
        AND #$10
        BNE OUTLOOP
        PLA
        STA $C090
        RTS             ; fertig
```

oder

```
OUTPUT  PHA             ; retten
OUTLOOP LDA $C099
        AND #$70        ; #$30 für Apple IIc
        CMP #$10
        BNE OUTLOOP     ; Super Serial Card
        PLA
        STA $C098
        RTS
```

Vor dem eigentlichen Druckzwischenspeicher liegt noch ein weiterer kleiner Speicher, der die Werte zur Initialisierung des Druckers aufnimmt. Dem EPSON muß vor **jeder** Druckzeile folgende Sequenz geschickt werden:
ESCAPE * 4 (Zeichenanzahl in 2 Bytes).
Bei einem anderen Fabrikat müssen Sie diese Werte nach den Angaben im Handbuch anpassen. „ESCAPE * 4" steht fest in diesem Puffer. Die Anzahl der Zeichen kann aber schwanken, da nicht alle Zeilen ganz gefüllt sind. Deshalb stehen auch hier zunächst nur zwei FF FF als Stellvertreter. Vom Programm wird für jede Druckzeile die Byteanzahl eingesetzt (Z. 130-136). In DRUCKEN wird dann dieser Code vor dem Druckzwischenspeicher gesendet.
In CLEAR wird der Druckzwischenspeicher auf Null gesetzt und die Anfangsadresse des Puffers wieder nach ABLAGE geschrieben, damit die nächsten Zeichen wieder von vorne beginnend abgelegt werden.
An CLEAR können wir sehen, daß es keine allgemeingültigen Empfehlungen für kurze und schnelle Assembler-Programme geben kann. Wir haben in diesem Programm an zwei Stellen selbstmodifizierenden Code benutzt, um knapp und schnell unser Ziel zu erreichen. Die Aufgabe von CLEAR ist es nun, $280 (= 640) hintereinanderliegende Bytes im Speicher zu löschen. Die Schwierigkeit liegt darin, daß sich die Anfangsadresse erst während der Assemblierung durch

die Länge des bis dorthin stehenden Codes ergibt. Unsere Routine sollte also mit jeder Startadresse laufen. Versuchen wir zunächst eine Lösung mit selbstmodifizierendem Code:

CLEAR1

```
                        1   ;*******************************
                        2   ; Löschprogramm für $280 Bytes *
                        3   ;*******************************
                        4   ;
                        5   ; Variante 1 mit selbstmodifizierendem
                        6   ; Code. Zeitbedarf: 16693 Prozessortakte
                        7   ;
                        8   PUFFER    EQU $1234       ;willkürlich
                        9   ;
                        10  ;
                        11            ORG $300
                        12  ;
                        13  ; Code initialisieren
                        14  ;
0300: D8                15  START     CLD
0301: A9 34             16            LDA #<PUFFER
0303: 8D 0E 03          17            STA LÖSCH+1
0306: A9 12             18            LDA #>PUFFER
0308: 8D 0F 03          19            STA LÖSCH+2
                        20  ;
                        21  ; Löschen
                        22  ;
030B: A9 00             23  CLEAR     LDA #$00
030D: 8D FF FF          24  LÖSCH     STA $FFFF       ;Dummy
                        25  ;
                        26  ; Adresse weitersetzen
                        27  ;
0310: EE 0E 03          28            INC LÖSCH+1     ;Lo
0313: D0 03             29            BNE TEST        ;Übertrag?
0315: EE 0F 03          30            INC LÖSCH+2     ;Ja, auch Hi
                        31  ;
                        32  ; Bereichsende testen
                        33  ;
0318: AD 0E 03          34  TEST      LDA LÖSCH+1     ;Lo-Byte
031B: CD 29 03          35            CMP PUFFENDE    ;gleich?
031E: D0 EB             36            BNE CLEAR       ;Nein, weiter
0320: AD 0F 03          37            LDA LÖSCH+2     ;Hi-Byte
0323: CD 2A 03          38            CMP PUFFENDE+1  ;gleich?
0326: D0 E3             39            BNE CLEAR       ;Nein, weiter
0328: 60                40            RTS             ;Ja, Ende
                        41  ;
0329: B4 14             42  PUFFENDE  ADR PUFFER+$280 ;Puffergröße
```

Zu Demonstrationszwecken wurde der Puffer willkürlich nach $1234 gelegt. Seine Anfangsadresse wird vom Programm in die Zeile 24 geschrieben, bevor die adressierte Speicherstelle auf Null gesetzt wird. Das Weitersetzen der Adresse und die Prüfung, ob das Ende des Puffers erreicht sind, geschehen ganz ähnlich wie in den Zeilen 114 bis 123 des Druckerprogramms.

Das Löschprogramm arbeitet ordentlich. Trotzdem habe ich es nicht benutzt, denn es ist mir zu langsam. Es benötigt 16693 Prozessortakte, um 640 Bytes zu löschen. Es geht viel Zeit damit verloren, daß nach jedem Byte die Adresse mit INC weitergesetzt und anschließend überprüft wird. Zudem geht dabei der Akkumulator verloren und muß stets mit LDA #$00 neu geladen werden. Der Vorteil dieser Routine ist, daß sie verhältnismäßig einfach zu verstehen ist.
Unser zweites Programm benötigt nur noch 7084 Prozessortakte, ist also mehr als doppelt so schnell:

CLEAR2

```
                    1   ;*******************************
                    2   ; Löschprogramm für $280 Bytes *
                    3   ;*******************************
                    4   ;
                    5   ; Variante 2 mit Ausnutzung aller Regi-
                    6   ; ster. Laufzeit: 7084 Prozessortakte
                    7   ;
                    8   PTR       EQU $0006      ;+$0007
                    9   PUFFER    EQU $1234      ;willkürlich
                   10   ;
                   11   ;
                   12             ORG $300
                   13   ;
                   14   ; 3 Schleifendurchgänge, in denen 1 x
                   15   ; $80 und 2 x $100 Bytes gelöscht werden.
                   16   ;
0300: D8           17   START     CLD
0301: A9 B4        18             LDA #<PUFFER-$80
0303: 85 06        19             STA PTR
0305: A9 11        20             LDA #>PUFFER-$80
0307: 85 07        21             STA PTR+1
0309: A2 03        22             LDX #$03
030B: A0 80        23             LDY #$80
030D: A9 00        24             LDA #$00
030F: 91 06        25   CLEAR     STA (PTR),Y
0311: C8           26             INY
0312: D0 FB        27             BNE CLEAR
0314: E6 07        28             INC PTR+1
0316: CA           29             DEX
0317: D0 F6        30             BNE CLEAR
0319: 60           31             RTS
```

Dieses Programm ist aus drei Gründen so schnell:
1. Der Akku wird nur einmal auf Null gesetzt
2. Das Lo-Byte wird mit INY hochgezählt und nicht mit INC
3. Die Bereichsüberprüfung geschieht durch eine Schleife.

Zu Punkt 1 ist wenig zu sagen. Zu Punkt 2: ein „INY" benötigt 2 Prozessortakte, ein „INC absolut" dagegen 6 Takte. Die Ausnutzung der indirekten indizierten Adressierung (STA (),Y) kostet zwar wieder 2 Takte mehr als ein einfaches

„STA absolut", es verbleiben aber 2 Takte auf der Habenseite. Der entscheidende Punkt ist der dritte: wir nutzen die Kenntnis aus, daß die genaue Anzahl der zu löschenden Bytes bekannt und konstant ist. Mit dem X-Register wird eine äußere Schleife aufgebaut, die dreimal durchlaufen wird. Darin eingebettet liegt die innere Schleife, die das Y-Register benutzt. Wenn wir nur die Zeilen 25 bis 27 sehen, so erkennen wir, daß das Y-Register bis $FF hochgezählt wird und die damit indizierten Speicherstellen nullgesetzt werden. Wenn wir Y mit $00 beginnen lassen, werden dadurch $100 Bytes ($00 – $FF) gelöscht. Anschließend wird das Hi-Byte weitergesetzt, damit der nächste Byteblock gelöscht werden kann. Da die äußere Schleife dreimal läuft, würden 3 * $100 Bytes gelöscht. Wir wollen aber nur $280 Bytes löschen. Wir zerlegen nun $280 in 2 * $100 + $80 Bytes. Die 2 * $100 Bytes lassen wir „hinten" in der Schleife laufen. Die $80 Bytes erreichen wir, indem wir Y beim ersten Mal nicht mit $00 starten lassen, sondern für den ersten Gang durch die äußere Schleife schon auf $80 hochsetzen (Zeile 23). Da die innnere Schleife immer mit Y = $00 endet, werden anschließend zweimal automatisch $100 Bytes gelöscht.
Eine kleine Korrektur ist noch vonnöten. Da das Y-Register zu Beginn bereits auf $80 steht, wir aber ab dem ersten Byte unseres Speichers löschen müssen, muß die auf die Nullseiten-Zeiger (PTR) gelegte Zieladresse um diese $80 zu niedrig angegeben werden. Der Assembler rechnet in Zeile 18 und 20 die korrigierte Adresse für uns aus.
Etwas Nachdenken und die Einbeziehung aller Register des 6502 hat so zu einer sehr kurzen und schnellen Routine geführt, die deshalb auch im Druckerprogramm benutzt wurde.

Die drei verschiedenen Anzeigearten des Apple (Text, LORES und HIRES) haben wir jetzt ausreichend kennengelernt. Diese Kapitel waren nicht ganz einfach. Wenden wir uns zur Entspannung jetzt den musikalischen Fähigkeiten des Apple zu.

4. Da steckt MUSIC drin

Der eingebaute Lautsprecher des Apple dient nicht nur zur Erzeugung des nervtötenden „Beep" bei jedem SYNTAX ERROR oder anderen schlimmen Nachrichten. Wir können dem Apple auch angenehmere Töne entlocken. Verglichen mit anderen Rechnern, die zum Teil eigene Schaltkreise für die Tonerzeugung über mehrere Oktaven besitzen, steht der Apple ziemlich nackt da. Wie wir schon im ersten Band gesehen haben, besteht die einzige Möglichkeit zur Lautäußerung darin, daß der Prozessor die Speicherstelle $C030 liest oder irgendetwas dort hineinschreibt. Die richtige Anzahl von erzeugten „Klicks" und ihr zeitlicher Abstand erzeugen einen hörbaren Ton, der uns mehr oder zumeist minder musikalisch erscheint.
Die Höhe eines Tones hängt von der Anzahl der Schwingungen pro Sekunde ab. Der Kammerton A hat z.B. 440 Schwingungen pro Sekunde oder 440 Hz. Je mehr Schwingungen pro Sekunde erfolgen, desto höher ist der erzeugte Ton. Da eine Schleife in Assembler sehr schnell ist, können wir sehr hohe Töne erzeugen, genau genommen sogar zu hohe. Die absolut kürzeste Form

```
0300: LDA $C030
0303: JMP $0300
```

benötigt 7 Prozessortakte. Da ca. 1 Million Takte pro Sekunde möglich sind, kommen wir auf ca. 142000 Schaltvorgänge in der Sekunde. So schnell kann sich die Membran des Lautsprechers nicht bewegen. Außerdem liegt die Tonhöhe in einem Bereich, der höchstens noch Fledermäusen zugänglich ist. Die Hörgrenze des Menschen liegt in jungen Jahren bei ca. 20000 Schwingungen/Sekunde. Um in den hörbaren Bereich zu gelangen, müssen wir also Warteschleifen einbauen. Die „Tasten-Musik" aus Band 1 ist ein Beispiel für die Programmierung solcher Verzögerungen. Wenn wir unser Assembler-Programm in Applesoft BASIC umsetzen, erhalten wir

```
10 A = PEEK (49200)
20 GOTO 10
```

Der resultierende Ton liegt mit ca. 70 Schwingungen/Sekunde im Baßbereich. BASIC ist daher für Musikprogramme denkbar ungeeignet. Der Ausweg aus dem Geschwindigkeitsproblem findet sich in einer Assembler-Routine, die wir auch von BASIC aus aufrufen können.
Nur die Wenigsten wissen, daß der Apple über ein eingebautes Assembler-Programm zur Musikerzeugung verfügt, das von Gary Shannon stammt. Es ist Bestandteil des „PROGRAMMERS AID NO.1" ROMs. Dieses stammt noch aus den Zeiten des „INTEGER BASIC" und ist heute als ROM in keinem Apple mehr enthalten. Aber jeder File „INTBASIC" auf der DOS 3.3 Sytem Master Diskette enthält auch das komplette „PROGRAMMERS AID", das bei jedem „LOADING INTEGER INTO LANGUAGE CARD" mit ins RAM gelangt, wenn Sie über wenigstens 64K Speicher verfügen. Die Dokumentation über INTEGER BASIC und das PROGRAMMERS AID wird von Apple für heutige Rechner nicht mehr mitgeliefert (überhaupt steht in den bunter werden den Heftchen immer weniger drin). Für alle, die dieses Musikprogramm benutzen wollen, habe ich es disassembliert und analysiert. Sie können es auch im normalen RAM-Bereich laufen lassen, wenn Sie über keine Language-Card (16K-Karte) verfügen oder diese durch ProDOS belegt ist. Zusammen mit ProDOS können Sie INTBASIC nicht benutzen, da auch ProDOS die Language-Card für sich benötigt. Unter ProDOS sollten Sie das Programm aber in einen geschützten Speicherbereich laden, wie das in Kapitel 10 erklärt ist.

MUSIC

```
1    ;*******************************
2    ; MUSIC Programmers Aid No. #1 *
3    ; „Integer-ROM" $D717 - $D7FB  *
4    ;*******************************
5    ;
6    ; Analysiert von Dr. Jürgen B. Kehrel
7    ; Für Verwendung ohne Language-Card um-
8    ; geschrieben auf normalen RAM-Bereich
9    ;
10   ;
11             ORG $9517      ;38167
12   ;
13   ; Zero-Page Zähler
14   ;
15   AB        EQU $00
16   AUF       EQU $01
17   LÄNGE     EQU $02         ;$02,$03
18   ;
19   ; Parameterübergabe
20   ;
21   KLANG     EQU $02FD
22   DAUER     EQU $02FE
23   FARBE     EQU $02FF
24   ;
25   ; I/O-Bereich
```

```
                        26  ;
                        27  SPKR      EQU $C030
                        28  ;
                        29  ; Spielt Note. Zyklus-Daten in „AUF" und „AB".
                        30  ; Zyklusdauer in „LÄNGE„. Zyklus zweigeteilt.
                        31  ;
9517: 4C 4E 95          32  START     JMP SUCHE       ;Best. Zyklus-Werte
                        33  ;
951A: A4 01             34  HOCH      LDY AUF         ;Pulsbr. „aufwärts„
951C: AD 30 C0          35            LDA SPKR        ;Lautsprecher
951F: E6 02             36  HOCH2     INC LÄNGE       ;Zyklusdauer
9521: D0 05             37            BNE HOCH3       ;noch nicht zu Ende
9523: E6 03             38            INC LÄNGE+1     ;Zyklusdauer
9525: D0 05             39            BNE HOCH4       ;verringere Pulsbr.
9527: 60                40            RTS             ;Zyklus abgelaufen
9528: EA                41  HOCH3     NOP             ; 5 Prozessortakte
9529: 4C 2C 95          42            JMP HOCH4       ; Zeitverzögerung
952C: 88                43  HOCH4     DEY             ;verringere Pulsbr.
952D: F0 05             44            BEQ RUNTER      ;Breite 0, „abwärts"
952F: 4C 32 95          45            JMP HOCH5       ; weitere 6 Prozes-
9532: D0 EB             46  HOCH5     BNE HOCH2       ; sortakte Zeitverz.
                        47  ;
9534: A4 00             48  RUNTER    LDY AB          ;Pulsbr. „abwärts„
9536: AD 30 C0          49            LDA SPKR        ;Lautsprecher
9539: E6 02             50  RUNTER1   INC LÄNGE       ;Zyklusdauer
953B: D0 05             51            BNE RUNTER2     ;noch nicht zu Ende
953D: E6 03             52            INC LÄNGE+1     ;Zyklusdauer
953F: D0 05             53            BNE RUNTER3     ;verringere Pulsbr.
9541: 60                54            RTS             ;Zyklus abgelaufen
9542: EA                55  RUNTER2   NOP             ; 5 Prozessortakte
9543: 4C 46 95          56            JMP RUNTER3     ; Zeitverzögerung
9546: 88                57  RUNTER3   DEY             ;verringere Pulsbr.
9547: F0 D1             58            BEQ HOCH        ;Breite 0, „aufwärts"
9549: 4C 4C 95          59            JMP RUNTER4     ; weitere 6 Prozes-
954C: D0 EB             60  RUNTER4   BNE RUNTER1    ; sortakte Zeitverz.
                        61  ;
                        62  ; Sucht in der Tabelle die richtigen Werte
                        63  ; von „AUF" und „AB" für die gegebene Note
                        64  ; in „FARBE" mit dem gewünschten „KLANG".
                        65  ;
954E: AD FF 02          66  SUCHE     LDA FARBE       ;gewünschte Note
9551: 0A                67            ASL             ;verdopple den Wert
9552: A8                68            TAY             ;nutze ihn als Index
9553: B9 96 95          69            LDA TABELLE,Y   ;suche Tabellenwert
9556: 85 00             70            STA AB          ;sp. ihn als „AB„
9558: AD FD 02          71            LDA KLANG       ;gewünschter Klang
955B: 4A                72  SHIFTEN   LSR             ;passe „AB„ an
955C: F0 04             73            BEQ NULL        ;durch „shiften",
955E: 46 00             74            LSR AB          ;bis ein Wert
9560: D0 F9             75            BNE SHIFTEN     ;Null geworden ist
9562: B9 96 95          76  NULL      LDA TABELLE,Y   ;hole Ursprungswert
9565: 38                77            SEC
9566: E5 00             78            SBC AB          ;berechne Differenz
9568: 85 01             79            STA AUF         ;sp. sie als „AUF"
```

```
956A: C8          80              INY             ;erhöhe den Index
956B: B9 96 95    81              LDA TABELLE,Y   ;hole nächsten Wert
956E: 65 00       82              ADC AB          ;addiere „AB„
9570: 85 00       83              STA AB          ;speichere Endwert
                  84    ;
9572: A9 00       85              LDA #$00        ;setze Akku auf Null
9574: 38          86              SEC             ;bilde Komplement
9575: ED FE 02    87              SBC DAUER       ;der gewünschten Dauer
9578: 85 03       88              STA LÄNGE+1     ;sp. als Hi-Byte
957A: A9 00       89              LDA #$00        ;setze das Lo-Bytex
957C: 85 02       90              STA LÄNGE       ;auf Null
957E: A5 01       91              LDA AUF         ;spielbare Note??
9580: D0 98       92              BNE HOCH        ;Ja, spiele sie
                  93    ;
                  94    ; Lege für die Note 00 eine Pause ein von der
                  95    ; selben Länge, als wenn sie gespielt würde
                  96    ;
9582: EA          97    PAUSE     NOP             ;zur Zeitverzögerung
9583: EA          98              NOP             ;7 Prozessortakte
9584: 4C 87 95    99              JMP PAUSE1
9587: E6 02       100   PAUSE1    INC LÄNGE       ;zähle Zyklusdauer
9589: D0 05       101             BNE PAUSE2
958B: E6 03       102             INC LÄNGE+1
958D: D0 05       103             BNE PAUSE3
958F: 60          104             RTS             ;Zyklus ist zu Ende
9590: EA          105   PAUSE2    NOP             ;weitere Prozessor-
9591: 4C 94 95    106             JMP PAUSE3      ;takte zur Zeitverz.
9594: D0 EC       107   PAUSE3    BNE PAUSE
                  108   ;
                  109   ; Tabelle mit den Notenwerten
                  110   ;
9596: 00 00 F6    111   TABELLE   HEX 0000F6F6E8E8DBDB
959E: CF CF C3    112             HEX CFCFC3C3B8B8AEAE
95A6: A4 A4 9B    113             HEX A4A49B9B92928A8A
95AE: 82 82 7B    114             HEX 82827B7B74746D6E
95B6: 67 68 61    115             HEX 676861625C5C5757
95BE: 52 52 4D    116             HEX 52524D4E49494545
95C6: 41 41 3D    117             HEX 41413D3E3A3A3637
95CE: 33 34 30    118             HEX 333430312E2E2B2C
95D6: 29 29 26    119             HEX 2929262724252223
95DE: 20 21 1E    120             HEX 20211E1F1D1D1B1C
95E6: 1A 1A 18    121             HEX 1A1A181917171516
95EE: 14 15 13    122             HEX 14151314121211 11
95F6: 10 10 0F    123             HEX 10100F100E0F
```

Damit Sie MUSIC auch gleich testen können, folgt ein kurzes DEMO-Programm:

MUSIC.DEMO

```
5   PRINT  CHR$ (4);"BLOAD MUSIC.OBJ"
10  MUSIC = 38167:KLANG = 765:DAUER = 766:FARBE = 767
20  POKE DAUER,40: POKE KLANG,32
30  FOR I = 1 TO 49
40  POKE FARBE,I
50  CALL MUSIC
60  NEXT I: END
```

Für KLANG können sie die Werte 2, 4, 8, 16, 32 und 64 wählen, wobei 32 der „Normalfall" ist. FARBE reicht von 00 bis 50, wobei 00 eine Pause darstellt. Statt FARBE können Sie auch NOTE sagen, aber nicht im BASIC-Programm benutzen, da Applesoft daraus NOT E macht. 32 entspricht ungefähr dem C'. Die Erhöhung von FARBE um 1 erhöht den Ton um einen Halbton. Sie können so Melodien über 4 Oktaven programmieren. Das BASIC-Programm poked die Werte für den nächsten Ton in die Speicherstellen 765 bis 767 ($02FD - $02FF), von wo sie durch das Maschinenprogramm abgeholt werden. Da die Parameterwerte nicht zerstört werden, müssen Sie nur die sich ändernden Werte jedesmal neu poken.
Da Tonleitern auf die Dauer etwas ermüdend sind, bereiten wir uns jetzt eine kleine Freude.

JOY

```
10  REM    JOY  FREI NACH BEETHOVEN
20  PRINT  CHR$ (4);"BLOAD MUSIC.OBJ"
30  MUSIC = 38167:KLANG = 765:DAUER = 766:FARBE = 767
40  RESTORE : POKE KLANG,32
50  FOR I = 1 TO 63: READ N
60  POKE DAUER,30 * (1 + (N > 300) + (N > 200) + (N > 100))
70  POKE FARBE,N - (100 * ((N > 300) + (N > 200) + (N > 100)))
80  CALL MUSIC: NEXT I
90  END
100 DATA  124,124,125,127,127,125,124,122,120,120
110 DATA  122,124,224,22,222,00,124,124,125,127,127
120 DATA  125,124,122,120,120,122,124,222,20,220,00
130 DATA  322,124,120,122,24,25,124,120
140 DATA  122,24,25,124,122,120,122,315,124,124
150 DATA  125,127,127,125,124,122,120,120
160 DATA  122,124,222,20,320
```

Die Noten sind in den DATA-Zeilen enthalten. Die beiden rechten Ziffern jeder Zahl geben den Notenwert an. Die linke der drei Ziffern bestimmt die Tonlänge: ist sie nicht vorhanden, ergibt das eine viertel Note; eine 1 bedingt eine halbe Note, eine 2 eine dreiviertel Note und eine 3 die ganze Note. In der Zeile 60 wird die Längen-Information umgesetzt. Die Zeile 70 reduziert den Wert N auf die letzten beiden Stellen, da der zulässige Bereich von 00 bis 50 reicht.

Wenn Sie möchten, können Sie MUSIC auch von einem Assembler-Programm aus aufrufen. Das folgende Beispiel soll ein startendes UFO darstellen. Die Tonfolge und die lange Laufzeit von ca. 65 Sekunden wird erzielt durch drei ineinander geschachtelte Schleifen, bei denen eine innere Schleife die Startwerte der Schleifenvariable aus der jeweils äußeren Schleife übernimmt.

UFO

```
                    1    ;**************************
                    2    ;   U.F.O Start mit MUSIC  *
                    3    ;**************************
                    4    ; Laufzeit ca. 65 Sekunden
                    5    ;
                    6              ORG $803
                    7    ;
                    8    ; Zählschleifen
                    9    S1        EQU $FD
                    10   S2        EQU $FE
                    11   S3        EQU $FF
                    12   ;
                    13   ; Aufruf von „MUSIC.OBJ" und Parameter
                    14   ;
                    15   MUSIC     EQU $9517
                    16   KLANG     EQU $2FD
                    17   DAUER     EQU $2FE
                    18   FARBE     EQU $2FF
                    19   ;
0803: A9 01         20   START     LDA #$01
0805: 8D FE 02      21             STA DAUER
0808: A9 10         22             LDA #$10
080A: 8D FD 02      23             STA KLANG
080D: A9 01         24   LOOP      LDA #$01       ;von $01 bis $7F
080F: 85 FD         25             STA S1
0811: A9 01         26   LOOP2     LDA #$01       ;von $01 bis $33
0813: 85 FE         27             STA S2
0815: 85 FF         28             STA S3
0817: A5 FF         29   LOOP3     LDA S3
0819: 8D FF 02      30             STA FARBE
081C: 20 17 95      31             JSR MUSIC      ;Ton
081F: 38            32             SEC
0820: A5 FF         33             LDA S3
0822: E5 FD         34             SBC S1
0824: 85 FF         35             STA S3
0826: B0 EF         36             BCS LOOP3
0828: E6 FE         37             INC S2
082A: A5 FE         38             LDA S2
082C: 85 FF         39             STA S3
082E: C9 33         40             CMP #$33
0830: 90 E5         41             BLT LOOP3
0832: E6 FD         42             INC S1
0834: 10 DB         43             BPL LOOP2
0836: 60            44             RTS
```

Vor dem Start von UFO muß von Ihnen „MUSIC.OBJ" ins RAM geladen werden.

Ihrer Phantasie sind für weitere Musikstücke keine Grenzen gesetzt, solange Ihnen ein einstimmiges Instrument ausreicht.

5. Der mobile Punkt

Im ersten Band haben wir uns schon ein wenig über Zahlenberechnungen unterhalten. Addition, Subtraktion, Multiplikation und Division sind Ihnen bekannt. Die betrachteten Zahlen hatte eine Gemeinsamkeit: es handelte sich immer um ganze Zahlen. Nun könnte Ihnen aber der Sinn danach stehen, 1.345 mit 2.4 zu multiplizieren oder die Quadratwurzel aus 7 zu berechnen. In BASIC ist das keine größere Schwierigkeit, in Assembler sind Sie aber einige Tage mit dem Programmieren beschäftigt, wenn Sie solche Aufgaben nicht schon einmal vorher gelöst haben. Was liegt da näher, als die eingebauten Routinen des Applesoft auch in Assembler-Programmen mitzubenutzen und sich dadurch viel Arbeit zu ersparen. Wir müssen lernen, mit dem Fließkomma-Paket des Applesoft-Interpreters umzugehen. Da im englischen Sprachgebiet für das Komma ein Punkt gesetzt wird, heißen diese Routinen dort „Floating-Point Subroutines". Wie es zu diesem Namen kommt, werden wir jetzt sehen. Unser erster Schritt besteht darin, die Darstellung einer Zahl im internen Applesoft-Format zu verstehen. Da der Prozessor mit nichts anderem umgehen kann als mit Folgen von Nullen und Einsen, muß es einen Weg geben, jede beliebige Zahl als Bitkette darzustellen.

5.1 Mit Geschenkpapier und Schleife: Einpacken und Auspacken

Ganze Zahlen werden zumeist in einem oder zwei Bytes dargestellt. Der Wertebereich geht dabei bis zu maximal 65535 ($FFFF). Wenn wir größere oder gebrochene Zahlen betrachten, benutzen wir dazu das Fließkomma-Format. Für den Anfänger ist dieses zumeist etwas schwierig zu verstehen. Wir werden es deshalb gemeinsam an einem Beispiel erarbeiten, für das wir die Zahl 123,828125 benutzen wollen (von jetzt an schreiben wir 123.828125).

Den Vorkomma-Anteil können wir leicht in eine Binärzahl umwandeln:
$123 = 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 1111011$
Hexadezimal erhalten wir den Wert $7B. Mit den Nachkomma-Stellen müssen wir genauso verfahren, nur daß wir diesmal die negativen Zweierpotenzen berücksichtigen müssen:
$.828125 = 1*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4} + 0*2^{-5} + 1*2^{-6} = .110101$
Setzen wir beides zusammen, erhalten wir:
123.828125 = % 1111011.110101

Wenn wir den Dezimalpunkt um eine Stelle nach links schieben, bedeutet das eine Division der Zahl durch 2. Soll der Zahlenwert konstant werden, müssen wir den neu erhaltenen Wert zum Ausgleich mit Potenzen von 2 multiplizieren. Die folgenden Ausdrücke sind alle gleichwertig:
$2^0 *$ % 1111011.110101
$2^1 *$ % 111101.1110101
$2^2 *$ % 11110.11110101
$2^3 *$ % 1111.011110101
$2^4 *$ % 111.1011110101
$2^5 *$ % 11.11011110101
$2^6 *$ % 1.111011110101
$2^7 *$ % .1111011110101

Die letzte Form, die vor dem Dezimalpunkt keine Stellen mehr aufweist, ist für uns besonders wichtig. Sie wird als die „normalisierte Form" der Zahl bezeichnet. Alle Zahlenumwandlungen in das Fließkomma-Format laufen über die normalisierte Form, bei der der Dezimalpunkt soweit verschoben wird, bis sich die am weitesten links befindliche 1 auf der rechten Seite des Punktes befindet. Wegen der Verschiebung des Kommas (des Punktes) werden solche Zahlen „Fließkomma-Zahlen" (Floating-Point-Numbers) genannt. Die Zahl 7 stellt den **Exponenten** und % .1111011110101 die **Mantisse** der Zahl dar. Wenn wir wollen, können wir die Mantisse auch hexadezimal schreiben. Wir trennen dazu ab dem Punkt von links nach rechts immer 4 Bits ab und rechnen sie zur HEX-Zahl um:
1111 = $F
0111 = $7
1010 = $A
1000 = $8
Das letzte unvollständige Nibble füllen wir mit Nullen auf. Wir bekommen damit
$123.828125 = 2^7 *$ %.1111011110101 $= 2^7 *$ $.F7A8

Applesoft benutzt zwei interne Darstellungsweisen für Fließkommazahlen. Betrachten wir zunächst die **ungepackte Form**, die immer 6 Bytes belegt. Ein Byte wird für den Exponenten benutzt, 4 Bytes für die Mantisse und 1 Byte für das Vorzeichen.

```
Exp.        Mantisse          Vorz.
-- -- -- -- -- -- -- -- -- --
```

Beim Exponenten weist Applesoft noch eine Besonderheit auf: zu dem eigentlichen Exponenten wird $80 addiert. Im Vorzeichen-Byte ist nur Bit 7 interessant: ist es gesetzt, ist die Zahl negativ, ist es gelöscht, so ist die Zahl positiv aufzufassen. Dies entspricht der uns bekannten Form einer vorzeichenbehafteten Ganzzahl.
Aus 123.828125 wird damit

```
Exp.        Mantisse          Vorz.
87 F7 A8 00 00 00 00
```

Die nicht benutzten Bytes der Mantisse werden mit Nullen gefüllt. Für die negative Zahl −123.828125 erhalten wir

```
Exp.        Mantisse          Vorz.
87 F7 A8 00 00 00 F7
```

Obwohl nur Bit 7 des Vorzeichen-Bytes von Bedeutung ist, setzt Applesoft hier meist das erste Mantissen-Byte mit entsprechend gesetztem Bit 7 ein.
Die ungepackte Form einer Fließkommazahl wird von Applesoft für alle internen Berechnungen benutzt. Zum Abspeichern ist die ungepackte Form jedoch recht platzverschwendend, da vom 6. Byte nur 1 Bit benötigt wird. Applesoft benutzt dazu die **gepackte Form**, die lediglich 5 Bytes benötigt. Die Umwandlung ist recht einfach:
Durch die Normalisierung ist das erste Bit rechts vom Dezimalpunkt eine 1. Da dies immer so ist (mit Ausnahme der Zahl Null), hat dieses Bit keinen eigentlichen Informationsgehalt, auch wenn es für Berechnungen benötigt wird. In der gepackten Form wird deshalb das Vorzeichenbit in Bit 7 des ersten Mantissen-Bytes aufbewahrt, d.h. es bleibt 1, wenn die Zahl negativ ist, es wird 0, wenn die Zahl positiv ist. Für Berechnungen wird daraus die ungepackte Form zurückgewonnen. (Jetzt können Sie sich auch erklären, warum bei negativen Zahlen das Vorzeichen-Byte in der ungepackten Form Teile des ersten Mantissen-Bytes enthält).
Wandeln wir die uns nun schon vertraute Zahl in die gepackte Form um:

```
Fließkomma-Zahl     ungepackte Form          gepackte Form
   123.828125   =  87 F7 A8 00 00 00   =   87 77 A8 00 00
  -123.828125   =  87 F7 A8 00 00 F7   =   87 F7 A8 00 00
```

Die Zahl Null paßt nicht in unser System, da sie nicht so normalisiert werden kann, daß rechts vom Punkt eine 1 steht. Applesoft benutzt für Null deshalb den Wert 00 00 00 00 00. Genaugenommen wird jede Fließkommadarstellung, die im Exponenten den Wert $00 enthält, von Applesoft als Null angesehen.

Ein weiteres Problem ist noch nicht gelöst: wie werden Zahlen dargestellt, deren Betrag zwischen 0 und 1 liegt. Betrachten wir 0.5. In der Exponentialschreibweise erhalten wir
$0.5 = 2^{-1} = 2^0 * \% .1$
Dieser Wert ist bereits normalisiert, so daß wir ihn leicht in die ungepackte Form umwandeln können:
0.5 = 80 80 00 00 00 00 (ungepackt)
Da 0.5 positiv ist, muß für die gepackte Form das Bit 7 des zweiten Bytes (1. Mantissen-Byte) gelöscht werden. Wir erhalten damit
0.5 = 80 00 00 00 00 (gepackt).
Mit dem negativen Wert ist es dann ganz einfach:
-0.5 = 80 80 00 00 00 80 (ungepackt)
-0.5 = 80 80 00 00 00 (gepackt).

Kommen wir zu einem zweiten Beispiel: 0.25.
$0.25 = 2^{-2} = 2^0 * .01$
Um diesen Wert zu normalisieren, müssen wir den Dezimalpunkt nach rechts verschieben. Zum Ausgleiche schreiben wir eine Zweierpotenz mit negativem Exponenten davor:
$2^0 * .01 = 2^{-1} * .1$
Um zum Exponenten der ungepackten Form zu gelangen, müssen wir $80 zu dem so gefundenen Exponenten addieren: -1 + $80 = $7F. Wir erhalten damit
.25 = 7F 80 00 00 00 00 (ungepackt)
.25 = 7F 00 00 00 00 (gepackt)
Wenden wir uns einem letzten Beispiel zu.
$.375 = 2^{-2} + 2^{-3} = 2^{-1} * .11$
.375 = 7F C0 00 00 00 00 (ungepackt)
.375 = 7F 40 00 00 00 (gepackt)

−.375 = 7F C0 00 00 00 C0 (ungepackt)
−.375 = 7F C0 00 00 00 (gepackt).

Zum Abschluß wollen wir noch die Frage beantworten, wie groß die größte positive Zahl ist, die Applesoft darstellt. Es ist
FF 7F FF FF FF = 1.70141183E+38

5.2 Alles ist fließend

Auf der Zero-Page sind für Applesoft zwei Speicherbereiche für ungepackte Fließkomma-Zahlen vorgesehen. Der Haupt-Fließkomma-Akkumulator (Main Floating-Point Accumulator, MFAC oder FAC abgekürzt) reicht von $009D bis $00A2. Der sekundäre Fließkomma-Akkumulator (Secondary Floating-Point Accumulator, SFAC oder ARG genannt) belegt die Speicherplätze von $00A5 bis $00AA.

MFAC		
Exp.	Mantisse	Vorz.
$009D	$009E – A1	$00A2
SFAC		
Exp.	Mantisse	Vorz.
$00A5	$00A6 – A9	$00AA

Daneben gibt es noch 2 Hilfs-Fließkomma-Akkumulatoren, die aber nur gelegentlich benötigt werden.

Die Bezeichnung „Akkumulator" wurde hier gewählt, da alle Fließkomma-Routinen mit diesen Speicherstellen rechnen, so daß sie für die Berechnungen eine ähnliche Bedeutung besitzen wie der Akkumulator des Prozessors für die einfachen Rechenoperationen des Prozessors. Der Fließkomma-Akkumulator hat aber nichts mit dem Prozessor-Akkumulator zu tun.

Im ROM des Apple sind einige Fließkomma-Zahlen verborgen, die der Applesoft-Interpreter für seine eigenen Berechnungen benötigt. Wir können sie mitbenutzen, wenn wir ihre Lage kennen. Darunter sind so nützliche Konstanten wie die Kreiszahl Pi oder der natürliche Logarithmus von 2.

Fließkomma-Zahlen im ROM

Wert	Startadresse	Inhalt
00	$E09A	00 00 (nur Exp./ 1. Mantisse)
−32768.0005	$E0FE	90 80 00 00 20
1	$E913	81 00 00 00 00
0.434255942	$E919	7F 5E 56 CB 79
0.576584541	$E91E	80 13 9B 0B 64
0.961800759	$E923	80 76 38 93 16
2.885390074	$E928	82 38 AA 3B 20
$\sqrt[2]{1/2}$	$E92D	80 35 04 F3 34
$\sqrt[2]{2}$	$E932	81 35 04 F3 34
−0.5	$E937	80 80 00 00 00
ln 2	$E93C	80 31 72 17 F8
−42.78203928	$EA46	86 AB 20 CE E7
10	$EA50	84 20 00 00 00
99999999.906	$ED0A	9B 3E BC 1F FD
999999999.25	$ED0F	9E 6E 6B 27 FD
10^9	$ED14	9E 6E 6B 28 00
0.5	$EE64	80 00 00 00 00
1.442695041	$EEDB	81 38 AA 3B 29 (1/ln 2)
$2.14987636*10^{-5}$	$EEE1	71 34 58 3E 56
$1.4352314*10^{-4}$	$EEE6	74 16 7E B3 1B
$1.34226348*10^{-3}$	$EEEB	77 2F EE E3 85
$9.61401701*10^{-3}$	$EEF0	7A 1D 84 1C 2A
$5.55051269*10^{-2}$	$EEF5	7C 63 59 58 0A
0.2402263846	$EEFA	7E 75 FD E7 C6
0.6931471862	$EEFF	80 31 72 18 10
1	$EF04	81 00 00 00 00
$1.18795464*10^{7}$	$EFA6	98 35 44 7A 68
$3.92767778*10^{-8}$	$EFAA	68 28 B1 46 20
$\pi/2$	$F066	81 49 0F DA A2
$2*\pi$	$F06B	83 49 0F DA A2
0.25	$F070	7F 00 00 00 00
−14.3813907	$F076	84 E6 1A 2D 1B
42.0077971	$F07B	86 28 07 FB F8
−76.7041703	$F080	87 99 68 89 01
81.6052237	$F085	87 23 35 DF E1
−41.3417021	$F08A	86 A5 5D E7 28
$2*\pi$	$F08F	83 49 0F DA A2

$-6.84793912*10^{-4}$	$F0CF	76 B3 83 BD D3
$4.85094216*10^{-3}$	$F0D4	79 1E F4 A6 F5
$-1.61117018*10^{-2}$	$F0D9	7B 83 FC B0 10
$3.42096380*10^{-2}$	$F0DE	7C 0C 1F 67 CA
$-5.42791328*10^{-2}$	$F0E3	7C DE 53 CB C1
$7.24571965*10^{-2}$	$F0E8	7D 14 64 70 4C
$-8.98023954*10^{-2}$	$F0ED	7D B7 EA 51 7A
0.110932413	$F0F2	7D 63 30 88 7E
−0.142839808	$F0F7	7E 92 44 99 3A
0.19999912	$F0FC	7E 4C CC 91 C7
−0.333333157	$F101	7F AA AA AA 13
1	$F106	81 00 00 00 00
0.811635157	$F123	80 4F C7 52 58

Die allermeisten dieser Zahlen werden Sie nicht benötigen, da es sich dabei um Reihenkoeffizienten für die internen Umrechnungen des Interpreters handelt. Sie sind aber der Vollständigkeit halber aufgeführt, damit Sie nicht zu lange knobeln müssen, wenn Sie sich einmal daran machen, Applesoft zu disassemblieren.

5.3 Das Zauberpaket wird entschnürt

Um Fließkomma-Zahlen im Speicher hin- und herzubewegen sowie zum Packen/Entpacken stehen zahlreiche ROM-Routinen zur Verfügung. Wir wollen die Möglichkeiten hier nur streifen und auf große Programme verzichten, da Mathematik-Programme zumeist sehr individuell geschrieben werden müssen. Zunächst erproben wir ein einfaches Multiplikations-Programm: 0.25 * 10. Wir benutzen dazu die gepackten Fließkommazahlen aus dem ROM und die Multiplikationsroutine FMULTT ($E982). FMULTT multipliziert den MFAC mit dem SFAC und legt das Ergebnis im MFAC ab. Dazu müssen wir unsere Zahlen entpacken und in die Fließkomma-Akkumulatoren schaffen. MOVFM ($EAF9) überträgt die angegebene Zahl aus dem Speicher (Memory) in den MFAC. Zusätzlich zum MFAC wird noch ein 5. Mantissen-Byte bei $00AC angelegt, das die Rechengenauigkeit erhöhen soll. Der Aufruf von FMULT ($E97F) überträgt die angegebene Zahl in den SFAC und ruft dann die eigentliche Multiplikationsroutine FMULTT ($E982) auf. Um die Lage einer gepackten Zahl im Speicher anzugeben, müssen wir den Akkumulator (des

Prozessors!) mit dem Lo-Byte und das Y-Register mit dem Hi-Byte ihrer Adresse laden, bevor wir die ROM-Routine aufrufen.

Eine Kleinigkeit ist noch zu beachten. Die Speicherstelle $00AB muß den Wert des Exclusive-OR verknüpften Vorzeichenbytes von MFAC und SFAC enthalten. Sowohl MOVFM als auch FMULT erledigen dies für uns mit.

FP-DEMO

```
                        1    ;**********************
                        2    ; FP - Demonstration *
                        3    ;**********************
                        4    ;
                        5    DOSWRM     EQU $03D0
                        6    ZEHN       EQU $EA50
                        7    VIERTEL    EQU $F070
                        8    ;
                        9    MFAC       EQU $009D
                        10   FMULT      EQU $E97F
                        11   MOVFM      EQU $EAF9
                        12   MOVMF      EQU $EB2B
                        13   SIN        EQU $EFF1
                        14   ;
                        15   HOME       EQU $FC58
                        16   CROUT      EQU $FD8E
                        17   PRBYTE     EQU $FDDA
                        18   COUT       EQU $FDED
                        19   ;
                        20              ORG $0300
                        21   ;
0300: D8                22   START      CLD              ;Binärmodus
0301: 20 58 FC          23              JSR HOME
0304: A0 00             24              LDY #$00
0306: B9 50 EA          25   LOOP1      LDA ZEHN,Y       ;ausgeben
0309: 20 DA FD          26              JSR PRBYTE       ;HEX-Zahl
030C: A9 A0             27              LDA #" "         ;Leerzeichen
030E: 20 ED FD          28              JSR COUT
0311: C8                29              INY
0312: C0 05             30              CPY #$05
0314: 90 F0             31              BLT LOOP1
0316: 20 8E FD          32              JSR CROUT        ;‹CR›
0319: A9 50             33              LDA #‹ZEHN
031B: A0 EA             34              LDY #›ZEHN
031D: 20 F9 EA          35              JSR MOVFM        ;Mem -› MFAC
0320: 20 70 03          36              JSR SHOWM        ;anzeigen
0323: 20 8E FD          37              JSR CROUT        ;‹CR›
0326: A0 00             38              LDY #$00
0328: B9 70 F0          39   LOOP2      LDA VIERTEL,Y    ;ausgeben
032B: 20 DA FD          40              JSR PRBYTE       ;HEX-Zahl
032E: A9 A0             41              LDA #" "         ;Leerzeichen
0330: 20 ED FD          42              JSR COUT
0333: C8                43              INY
0334: C0 05             44              CPY #$05
0336: 90 F0             45              BLT LOOP2
```

```
0338: 20 8E FD  46              JSR CROUT       ;‹CR›
033B: 20 8E FD  47              JSR CROUT
033E: A9 70     48              LDA #‹VIERTEL
0340: A0 F0     49              LDY #›VIERTEL
0342: 20 7F E9  50              JSR FMULT       ;Mem -› SFAC, dann
0345: 20 8E FD  51              JSR CROUT       ;multiplizieren
0348: 20 70 03  52              JSR SHOWM       ;anzeigen
034B: A2 6B     53              LDX #‹SPEICHER
034D: A0 03     54              LDY #›SPEICHER
034F: 20 2B EB  55              JSR MOVMF       ;MFAC -› Mem
0352: 20 85 03  56              JSR SHOWSP      ;Speicher anzeigen
0355: 20 8E FD  57              JSR CROUT
0358: 20 F1 EF  58              JSR SIN         ;Sinus berechnen
035B: 20 70 03  59              JSR SHOWM       ;MFAC zeigen
035E: A2 6B     60              LDX #‹SPEICHER
0360: A0 03     61              LDY #›SPEICHER
0362: 20 2B EB  62              JSR MOVMF       ;MFAC -› Speicher
0365: 20 85 03  63              JSR SHOWSP
0368: 4C D0 03  64              JMP DOSWRM      ;Warmstart über DOS
                65  ;
                66  SPEICHER   DFS 5           ;Speicherplatz
                67  ;
0370: A0 00     68  SHOWM       LDY #$00        ;MFAC anzeigen
0372: B9 9D 00  69  LOOP3       LDA MFAC,Y
0375: 20 DA FD  70              JSR PRBYTE
0378: A9 A0     71              LDA #" "
037A: 20 ED FD  72              JSR COUT
037D: C8        73              INY
037E: C0 06     74              CPY #$06
0380: 90 F0     75              BLT LOOP3
0382: 4C 8E FD  76              JMP CROUT
                77  ;
0385: A0 00     78  SHOWSP      LDY #$00        ;Speicher anzeigen
0387: B9 6B 03  79  LOOP4       LDA SPEICHER,Y  ;anzeigen
038A: 20 DA FD  80              JSR PRBYTE      ;HEX-Zahl
038D: A9 A0     81              LDA #" "        ;Leerzeichen
038F: 20 ED FD  82              JSR COUT
0392: C8        83              INY
0393: C0 05     84              CPY #$05
0395: 90 F0     85              BLT LOOP4
0397: 4C 8E FD  86              JMP CROUT       ;‹CR›, dann zurück
```

Das Programm benutzt zwei Monitor-Routinen, die wir bisher noch nicht behandelt haben. CROUT ($FD8E) gibt ein „Carriage Return“ ($8D) über COUT aus. PRBYTE ($FDDA) gibt das Byte im Akkumulator über COUT als 2-ziffrige HEX-Zahl aus.

CROUT $FD8E
Gibt $8D über COUT aus.

Eingabe: –

Ausgabe: Akku = $8D

PRBYTE $FDDA
Gibt Akkumulator als ASCII-HEX-Zahl über COUT aus

Eingabe: Akku = HEX-Zahl

Ausgabe: Akku zerstört!

Nach dem Löschen des Bildschirms sehen Sie in der ersten Zeile die gepackte Zahl 10, darunter die ungepackte Zahl, so wie sie im MFAC steht. Es folgt die gepackte Zahl 1/4. Das Endergebnis der Multiplikation (2.5) steht wiederum ungepackt im MFAC und wird darunter ausgegeben. Schließlich wird das Ergebnis noch in den Speicher kopiert und dabei erneut gepackt (MOVMF $EB2B).

```
84 20 00 00 00       10 gepackt
84 A0 00 00 00 20    10 ungepackt

7F 00 00 00 00       1/4 gepackt


82 A0 00 00 00 20    2.5 ungepackt
82 20 00 00 00       2.5 gepackt

80 99 35 78 6E 18    SIN 2.5 ungepackt
80 19 35 78 6E       SIN 2.5 gepackt
```

Applesoft bietet einen ganzen Satz von Funktionen an, aus dem wir stellvertretend die Sinus-Funktion benutzen. Alle Funktionen wirken auf den MFAC und werden genau wie SIN ($EFF1) aufgerufen. Das Rechenergebnis findet sich wieder im MFAC. Auch den Sinus von 2.5 sehen wir uns ungepackt und dann im SPEICHER gepackt an. Dabei können Sie feststellen, daß die ungepackte Form im Vorzeichenbyte gelegentlich auch abweichende Werte aus Zwischenergebnissen enthalten kann. Von Bedeutung ist nur Bit 7.

FP-Operationen mit einem Operanden

Alle folgenden Routinen beziehen sich auf den MFAC, der vorher entsprechend geladen werden muß.

NOT $DE98 negiert MFAC
MFAC = 00 ⇒ MFAC = 1
MFAC ≠ 00 ⇒ MFAC = 0

FALSE $DF5D trägt 0 im MFAC ein (nur $009D!)

TRUE $DF60 trägt 1 im MFAC ein

FADDH $E7A0 addiert 0.5 zum MFAC

NORM $E82E normalisiert MFAC mit 5. Mantissen-Byte

ZEROFAC $E84E setzt MFAC auf Null ($009D und $00A2)

LOG $E941 natürlicher Logarithmus vom MFAC

MULT10 $EA39 multipliziert MFAC mit 10

DIV10 $EA55 dividiert MFAC durch 10

RNDB $EB72 das 5. Mantissen-Byte wird nach MFAC gerundet

SGNA $EB82 Vorzeichen von MFAC testen.
MFAC < 0 ⇒ Akku = $FF
MFAC = 0 ⇒ Akku = $00
MFAC > 0 ⇒ Akku = $01

SGN $EB90 trägt das Vorzeichen vom MFAC im MFAC ein (der alte Inhalt wird zerstört!)
MFAC < 0 ⇒ MFAC = -1
MFAC = 0 ⇒ MFAC = 0
MFAC > 0 ⇒ MFAC = 1

ABS $EBAF bildet den Betrag vom MFAC

INT $EC23 isoliert den ganzzahligen Anteil

SQR $EE8D berechnet die Quadratwurzel vom MFAC

NEGOP $EED0 bildet den negativen Wert vom MFAC

EXP $EF09 exponiert MFAC zur Basis e (2.718...)

RND $EFAE berechnet eine „Zufallszahl" in Abhängigkeit von MFAC. In $00C9-CD muß vorher ein Startwert stehen. MFAC und $00C9-CD enthalten neue Zufallszahl.

COS $EFEA berechnet Cosinus vom MFAC

SIN $EFF1 berechnet Sinus vom MFAC

TAN $F03A berechnet Tangens vom MFAC

ATN $F09E berechnet Arcustangens vom MFAC

Die folgenden Routinen benötigen 2 Operanden, von denen einer im MFAC und der andere im SFAC stehen muß. Mit Ausnahme der Potenzfunktion haben alle Routinen zwei Eintrittspunkte: der erste nimmt an, daß MFAC schon geladen ist, und überträgt vor der eigentlichen Operation die gepackte Fließkomma-Zahl, auf die der Akkumulator und das Y-Register (Lo/Hi) zeigen, nach SFAC. Der zweite Eintrittspunkt erwartet, daß MFAC und SFAC geladen sind, und führt sofort die Operation aus. Vor dem Aufruf muß der Exponent von MFAC in den Akkumulator geladen werden (LDA $9D)!! Das Ergebnis findet sich immer im MFAC.

FP-Operationen mit zwei Operanden

FSUB	$E7A7 überträgt gepackte Zahl (A,Y) nach SFAC und führt dann FSUBT aus
FSUBT	$E7AA subtrahiert MFAC von SFAC (SFAC − MFAC)
FADD	$E7BE überträgt gepackte Zahl (A,Y) nach SFAC und führt dann FADDT aus
FADDT	$E7C1 addiert SFAC und MFAC (SFAC + MFAC)
FMULT	$E97F überträgt gepackte Zahl (A,Y) nach SFAC und führt dann FMULTT aus
FMULTT	$E982 multipliziert SFAC und MFAC (SFAC * MFAC)
FDIV	$EA66 überträgt gepackte Zahl (A,Y) nach SFAC und führt dann FDIVT aus
FDIVT	$EA69 dividiert SFAC durch MFAC (SFAC/MFAC)
FPWRT	$EE97 potenziert SFAC mit MFAC (SFAC ↑ MFAC)

Es folgen noch einige weitere Routinen, die nicht in ein Schema passen.

Logische Operationen und Vergleiche

NOT	$DE98 negiert MFAC MFAC = 00 ⇒ MFAC = 1 MFAC ≠ 00 ⇒ MFAC = 0
OR	$DF4F verknüpft SFAC und MFAC durch logisches ODER. Ist ein Operand ≠ 0, wird MFAC = 1, sonst ist MFAC = 0.
AND	$DF55 verknüpft SFAC und MFAC durch logisches UND. Sind beide Operanden ≠ 0, wird MFAC = 1, sonst ist MFAC = 0

COMP $DF6A vergleicht SFAC mit MFAC. MFAC wird auf 1 gesetzt, wenn der Vergleich wahr ist, andernfalls auf 0. Die Speicherstelle $0016 bestimmt den Typ des Vergleichs:

$0016	Vergleich	< = >
1	SFAC > MFAC	
2	SFAC = MFAC	
3	SFAC ≧ MFAC	
4	SFAC < MFAC	4 2 1
5	SFAC ≠ MFAC	
6	SFAC ≦ MFAC	

FCOMP $EBB2 subtrahiert den MFAC von der gepackten Zahl, auf die der Akkumulator und das Y-Register (Lo/Hi) weisen. Setzt den Akkumulator entsprechend dem Ergebnis. Zusätzlich wird die Nullflagge im Statusregister gesetzt, wenn beide Zahlen übereinstimmen.
MFAC < Zahl ⇒ Akku = $FF
MFAC = Zahl ⇒ Akku = $00
MFAC > Zahl ⇒ Akku = $01

5.4 Sich regen bringt Segen

Alle bisherigen Routinen gingen davon aus, daß sich die Fließkomma-Zahlen schon irgendwo im Speicher befinden. In einem Programm ist das zumeist aber nicht der Fall. Wir benötigen also weitere Möglichkeiten, um Zahlen innerhalb des Speichers zu bewegen und um Zahlen zwischen der Fließkomma-Darstellung und anderen Betrachtungsweisen zu wandeln, denn schließlich sind in 5 Bytes gepackte Zahlen nicht sehr anschaulich.
Die folgenden Routinen dienen dem Verschieben von Fließkomma-Zahlen und der Umwandlung zwischen gepackter und ungepackter Form. Der Akkumulator und das Y-Register zeigen gegebenenfalls auf das Lo- bzw. das Hi-Byte der Anfangsadresse (Exponentenadresse).

FP-Übertragungs-Routinen

MOVESM $E9E3 überträgt gepackte Zahl (A,Y) nach SFAC. Die Vorzeichen von MFAC und SFAC werden verknüpft und nach $00AB gespeichert.

MOVEFM $EAF9 überträgt gepackte Zahl (A,Y) nach MFAC, legt ein auf Null gesetztes 5. Mantissen-Byte an

MOVE2F $EB1E überträgt MFAC in den 2. Hilfsfließkomma-Akkumulator ($0098-9C)

MOVE1F $EB21 überträgt MFAC in den 1. Hilfsfließkomma-Akkumulator ($0093-97)

MOVEZF $EB23 überträgt MFAC in die Zero-Page Speicherstelle, auf die das X-Register weist

MOVEMF $EB2B rundet MFAC und überträgt ihn als gepackte Zahl nach (X,Y)!!!

MOVEFS $EB53 überträgt SFAC nach MFAC

MOVESF $EB63 überträgt MFAC nach SFAC

Die beiden wichtigsten Formen der Zahlendarstellung im Apple sind das Fließkomma-Format und das vorzeichenbehaftete Ganzzahl-Format (signed Integer). Eine letzte Klasse von Routinen des Applesoft-Interpreters dient nun zur Umrechnung (Konvertierung) zwischen diesen beiden Systemen. Ein Programmbeispiel folgt in Kapitel 7.

FP-Konvertierungs-Routinen

VARL $DED5 überträgt den Wert einer Variablen nach MFAC, wenn es sich um eine numerische Variable handelt. TXTPTR ($00B8/B9) muß auf das erste Zeichen des Variablennamens zeigen.

VARL1 $DEE9 VPNT ($00A0/A1) enthält die Startadresse der 2-Byte-Integerzahl mit Vorzeichen, die nach MFAC übertragen wird.

MKINT $E108 verwandelt den MFAC in eine 2-Byte-Integerzahl in VPNT ($00A0 = Hi/$00A1 = Lo!!). MFAC muß positiv und < 32768 sein, sonst erfolgt Fehlermeldung.

AYINT $E10C verwandelt den MFAC in eine 2-Byte-Integerzahl mit Vorzeichen in VPNT ($00A0/A1 = Hi/Lo!). MFAC muß zwischen −32768 und +32768 liegen, sonst erfolgt Fehlermeldung.

GIVAYF $E2F2 verwandelt eine vorzeichenbehaftete 2-Byte-Integerzahl (Y-Reg = Lo, Akku = Hi) in eine Fließkomma-Zahl in MFAC

SGNFLT $E301 die 1-Byte-Integerzahl ohne(!) Vorzeichen im Y-Register wird in eine Fließkommazahl in MFAC umgewandelt. Der Akkumulator muß beim Aufruf Null sein (Akku = 00).

CONINT $E6FB verwandelt den MFAC in eine 1-Byte-Integerzahl im X-Register. Routine endet nicht mit RTS sondern mit JMP CHRGOT!

GETADR $E752 verwandelt den MFAC in eine 2-Byte-Integerzahl ohne Vorzeichen in LINNUM ($0050 = Lo, $0051 = Hi). Wertebereich 0 – 65535.

FLOAT $EB93 verwandelt die vorzeichenbehaftete 1-Byte-Integerzahl im Akkumulator in eine Fließkomma-Zahl im MFAC.

QUINT $EBF2 verwandelt MFAC in eine 4-Byte-Integerzahl mit Vorzeichen. Ergebnis wird von $009E bis $00A2 mit dem Vorzeichen in Bit 7 von $009E abgelegt.

FIN $EC4A verwandelt einen String in eine Fließkommazahl. TXTPTR muß vor das erste Zeichen zeigen. Zuerst „JSR CHRGET“ ausführen, dann „JSR FIN“. String muß mit gelöschtem Bit 7 im Speicher stehen und mit „:“, $00 oder Nichtzahl-Zeichen enden.

PRNTFAC $ED2E gibt den MFAC als Dezimalzahl über COUT aus. Benutzt FOUT, STROUT und COUT.

FOUT $ED34 verwandelt den MFAC in einen Zahlenstring ab $0100 um (im Stack). String hat Bit 7 gelöscht und $00 als Endmarker. Akku (Lo) und Y-Register (Hi) zeigen auf den Stringanfang.

Wenn Sie in einem Programm kurzfristig den MFAC retten müssen, können Sie die Routine PSHMFAC ($DE10) benutzen, um die Fließkomma-Zahl auf den Stack zu retten. Mit PULLSFAC ($DE47) kann sie von dort zurück in den SFAC geholt werden. Da beide Routinen den Stackpointer versetzen, können sie nicht einfach mit JSR aufgerufen werden. PSHMFAC zieht die Return-Adresse vom Stack und legt sie (nach Addition von 1) in den Speicherstellen $005E und $005F ab. Die Routine endet mit einem JMP ($5E). PULLSFAC endet mit einem RTS. Da vorher aber bereits die Fließkomma-Werte vom Stack geholt werden, muß **vor** der Fließkomma-Zahl bereits die Rücksprungadresse auf dem Stack liegen und PULLSFAC mit einem JMP aufgerufen werden.

PSHMFAC $DE10 schiebt MFAC von hinten mit dem Vorzeichen beginnend auf den Stack. Holt sich vorher seine Rücksprungadresse vom Stack und springt sie später mit JMP () an.

PULLSFAC $DE47 holt 6 Bytes vom Stack und legt sie im SFAC ab. Routine muß mit JMP aufgerufen werden (nicht JSR) und erwartet die Rücksprungadresse oberhalb der Zahl auf dem Stack.

Ein kurzes Beispiel soll die Lage durchsichtig machen:

```
        LDA #<WEITER
        PHA                 ;Returnadresse für PULLSFAC
        LDA #>WEITER
        PHA
        JSR PSHMFAC         ;MFAC -> STACK
        ......              ;Ihre Routinen
        JMP PULLSFAC        ;STACK -> SFAC
WEITER  ...                 ;hierhin kehrt PULLSFAC zurück
```

Ihre eigenen zwischengeschalteten Routinen müssen zwangsläufig den Stackpointer unverändert lassen, d.h. er darf zwischenzeitlich verändert werden, wenn er vor dem Aufruf von PULLSFAC wieder auf seinen alten Wert zurückgebracht wird.

Unser Fließkomma-Paket ist entschnürt. Aus Platzgründen können nicht für alle Routinen Anwendungen angegeben werden. Die verschiedenen Tabellen sind aber ausreichend, um ganze Mathematik-Programme mit reellen Zahlen in Assembler zu schreiben. Sie können alle Unterprogramme benutzen, auch wenn Sie nicht wissen, wie der Applesoft-Interpreter intern z.B. einen Logarithmus-Wert berechnet. Auch im nächsten Kapitel über Applesoft-Variablen werden wir einen Teil der Fließkomma-Routinen sinnvoll für uns nutzen.

6. Und immer schön variabel bleiben!

Applesoft hat eine weitere Eigenschaft, die wir als Assembler-Programmierer mitbenutzen können: die Variablen. Wenn wir Programme schreiben wollen, die sowohl aus BASIC als auch aus Teilen in Maschinensprache bestehen, ist eine Kenntnis über den Aufbau und die Verwaltung der Applesoft-Variablen unumgänglich. Nur wenn wir auch von der Maschinensprache auf Variablen zugreifen können, ist es einfach möglich, zwischen BASIC und Assembler Informationen auszutauschen. Alles andere führt zu einem zeit- und platzverschwendenden „herumpoken".

6.1 Eine Tabelle ist aller Variablen Anfang

Applesoft unterscheidet 3 Typen von einfachen Variablen, 3 Typen von Feldvariablen und einen Funktionstyp. Sie werden in BASIC durch die Schreibweise unterschieden:

Reelle Zahl:	nur Name z.B. A
Ganzzahl:	Name mit angehängtem „%" z.B. A%
Zeichenkette:	Name mit angehängtem „$" z.B. A$
Funktion:	Name mit vorangestelltem „FN" z.B. FN A(B)
Feld:	Name wie oben, mit angehängtem Index z.B. A(1), A%(1), A$(1)

Betrachten wir zunächst die einfachen Variablen.
Applesoft legt alle einfachen Variablen in einer Tabelle ab, die dem BASIC-Programm normalerweise unmittelbar folgt. Auf der Null-Seite ist der Zeiger VARTAB ($0069/6A) vorhanden, der immer auf den Beginn der Tabelle zeigt. Dieser Wert ist mit dem BASIC-LOMEM: identisch. Die Adresse des ersten

Bytes hinter der Tabelle ist in ARYTAB ($006B/6C) abgelegt. Jeder Eintrag in der Tabelle ist 7 Bytes lang, auch wenn nicht immer alle 7 Bytes benötigt werden.

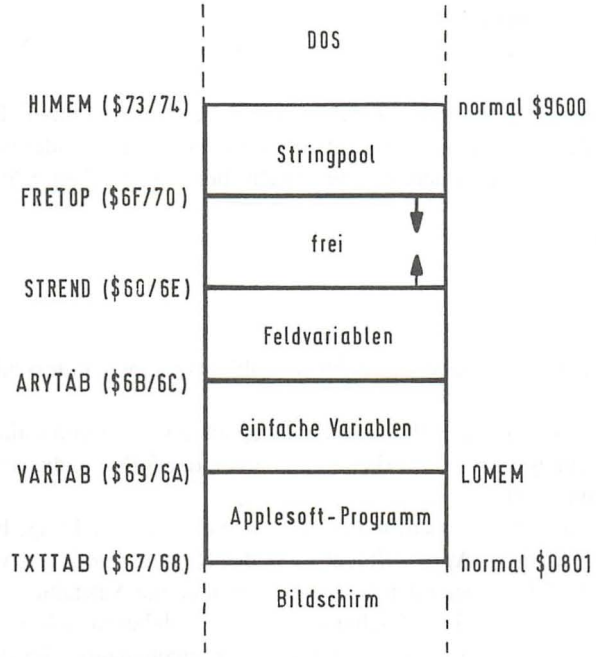


Abb. 8: Zeiger der Speicherbelegung

Die Aufteilung innerhalb der Tabelle geschieht durch den Zeitpunkt der Variablendefinition. Die zuerst definierte Variable steht an der untersten Position. Ihr folgt die als zweites definierte Variable usw. So kommt es, daß alle Typen durcheinanderstehen.
Die Funktionen finden sich ebenfalls in der selben Tabelle.

Applesoft braucht also ein Unterscheidungsmerkmal, um die 4 Möglichkeiten auseinander zu halten. Da sich Applesoft nur die ersten 2 Buchstaben von jeder Variablenbezeichnung merkt (HANS und HAUS bezeichnen für Applesoft also die gleiche Variable!), eignen sich die beiden Bit 7 von jedem Namen sehr gut zu

diesem Zweck. Ein gesetztes Bit 7 ergibt einen negativen Wert, ein gelöschtes Bit 7 einen positiven Wert. Wir erhalten damit:

Variablentyp	1. Zeichen	2. Zeichen
reelle Zahl	positiv 0	positiv 0
ganze Zahl	negativ 1	negativ 1
Zeichenkette	positiv 0	negativ 1
Funktion	negativ 1	positiv 0

Die ersten beiden Bytes jedes Tabelleneintrags enthalten immer die ersten beiden Bytes des Variablennamens mit entsprechend gesetzten oder gelöschten Bits 7. Ist die Variablenbezeichnung nur ein Zeichen lang, wird mit $00 bzw. $80 aufgefüllt.

A ⇒ $41 $00
A% ⇒ $C1 $80
A$ ⇒ $41 $80
FN A(X) ⇒ $C1 $00

Die restlichen 5 Bytes haben je nach Variablentyp einen unterschiedlichen Inhalt.
Bei den reellen Variablen enthalten sie die gepackte Fließkommazahl.
Die ganzen Zahlen belegen nur die ersten beiden der 5 Bytes, die restlichen 3 sind mit $00 aufgefüllt.
Die Zeichenketten passen nicht selber in die Tabelle, da ihre Länge bis zu 255 Zeichen betragen kann. Applesoft legt sie in der Reihenfolge Ihres Auftretens bei HIMEM ($0073/74) beginnend abwärts ab (auf einige Ausnahmen kommen wir später zu sprechen). Die Zeichenketten stehen dabei in sich aber wieder aufwärts. Das erste Byte der Tabelle enthält eine Längenangabe, das zweite und dritte Byte in der Tabelle zeigt auf das erste Byte der Zeichenkette. Die beiden letzten Bytes sind leer ($00). Wenn HIMEM bei $9600 liegt und „ABCDE" die erste definierte Zeichenkette ist, finden wir:

$9600 = ???? ⇐ HIMEM
$95FF = E
$95FE = D
$95FD = C
$95FC = B
$95FB = A ← hierhin zeigt der Tabelleneintrag

Das untere Ende des Zeichenkettenbereichs (Stringpool) wird in dem Zeiger FRETOP ($006F/70) festgehalten, da hier der nächste auftretende String abgelegt werden muß.

Die Tabellen wachsen also von unten (LOMEM:) nach oben, der Stringpool von oben (HIMEM:) nach unten. Treffen beide Bereiche aufeinander (STREND und FRETOP überkreuzen sich), ist der gesamte Speicher voll und Applesoft gibt eine „MEMORY FULL ERROR" aus, wenn sich durch eine Umorganisation des Speichers (Garbage Collection = interne Müllabfuhr) nicht noch wieder Platz gewinnen läßt.
Bei den Funktionen schließlich ist die Sache noch etwas komplizierter. Nach den beiden Namensbytes finden wir hier nämlich in 2 Bytes einen Zeiger auf die Adresse der Speicherstelle nach dem „="-Zeichen im BASIC-Programm. Die Funktionsdefinition wird von Applesoft nämlich bei jeder Benutzung aus dem BASIC-Text gelesen. Die nächsten beiden Bytes enthalten die Adresse des Wertes der Pseudovariablen, die bei der Definition in Klammern angegeben werden muß. Sie zeigen also auf eine 5-Byte-Gruppe innerhalb der selben Tabelle. Das letzte Byte enthält schließlich das erste Zeichen nach dem „="-Zeichen im BASIC-Text, ev. in tokenisierter Form. Dieses Byte wird aber an keiner Stelle ausgewertet.

Fassen wir unsere Erkenntnisse noch einmal in einer Tabelle zusammen:

Einfache Applesoft-Variablen

Reelle Zahl	Ganze Zahl	Zeichenkette	Funktion
Name 1. Byte 0 2. Byte 0	Name 1. Byte 1 2. Byte 1	Name 1. Byte 0 2. Byte 1	Name 1. Byte 1 2. Byte 0
Exponent	Wert Lo-Byte	Längen-Byte	Lo-Adr. nach =
Mantisse MSB	Wert Hi-Byte	Adresse Lo-Byte	Hi-Adr. nach =
Mantisse	00	Adresse Hi-Byte	Lo-Adr. Variab.
Mantisse	00	00	Hi-Adr. Variab.
Mantisse LSB	00	00	Zeichen nach =

Die Feldvariablen (auch Matrizen genannt) liegen oberhalb der einfachen Variablen. ARYTAB ($006B/6C) zeigt auf den Anfang der Feldvariablen, STREND ($006D/6E) auf das Ende. Die Feldvariablen werden genauso bezeichnet wie die einfachen Variablen, nur das hinter die Bezeichnung noch eine Klammer gesetzt wird. Diese enthält eine Index-Liste, die aus durch Komma getrennten numerischen Ausdrücken für jede Dimension besteht. Von jedem numerischen Ausdruck wird nur der ganzzahlige Anteil als Index ausgewertet (z.B. A$(1,2,1.1+4) = A$(1,2,5))
Die Länge der Eintragungen für Feldvariablen ist unterschiedlich. Ein Eintrag enthält zunächst die ersten beiden Bytes des Namens, wobei der Typ genauso

codiert ist wie bei den einfachen Variablen. Der Funktions-Typ entfällt.
Im 3. und 4. Byte steht ein sogenannter Offset-Zeiger. Er gibt an, wieviele Bytes zum Anfang dieser Tabelle addiert werden müssen, um zum Beginn der nächsten Tabelle zu gelangen. Diese Information ist wichtig, weil alle Tabellen unterschiedlich lang sein können. Das 5. Byte enthält die Zahl der Dimensionen. Für jede Dimension folgen jetzt 2 Bytes, die die Größe der Dimension angeben. Diese Bytepaare beginnen mit der am weitesten rechts stehenden Dimension (bei A(1,2,3) also mit 3).
Der soeben besprochene Teil der Tabelle wird auch Struktur- oder Informationsblock (= Variablen-Kopf) genannt. Er ist für alle Variablentypen gleich aufgebaut. Es schließt sich der Werteblock an, in dem eine reelle Zahl 5 Bytes, eine ganze Zahl 2 Bytes und eine Zeichenkette 3 Bytes belegt. Jeder Eintrag entspricht dem der einfachen Variablen. Im Gegensatz zu den einfachen Variablen gibt es bei den Feldern aber kein Auffüllen mit Nullen. Bei mehrdimensionalen Feldern sind die Werte im Werteblock so angeordnet, daß der am weitesten rechts stehende Index am langsamsten anwächst.
Fassen wir auch diesmal alle Informationen in einer Tabelle zusammen:

Applesoft Feldvariablen

Reelle Zahl	Ganze Zahl	Zeichenkette
Name 1. Byte 0	Name 1. Byte 1	Name 1. Byte 0
2. Byte 0	2. Byte 1	2. Byte 1
Offset Lo-Byte	Offset Lo-Byte	Offset Lo-Byte
Offset Hi-Byte	Offset Hi-Byte	Offset Hi-Byte
Dimensionen N	Dimensionen N	Dimensionen N
Lo-Größe Dim. N	Lo-Größe Dim. N	Lo-Größe Dim. N
Hi-Größe Dim. N	Hi-Größe Dim..N	Hi-Größe Dim. N
.	.	.
.	.	.
Lo-Größe Dim. 0	Lo-Größe Dim. 0	Lo-Größe Dim. 0
Hi-Größe Dim. 0	Hi-Größe Dim. 0	Hi-Größe Dim. 0
Wert von (0,0,..0)	Wert von (0,0,..0)	Stringzeiger (0,0,..0)
Exponent	Wert Lo-Byte	Längen-Byte
Mantisse MSB	Wert Hi-Byte	Adresse Lo-Byte
Mantisse	----------	Adresse Hi-Byte
Mantisse	Wert von (1,0,..0)	----------
Mantisse LSB	Wert Lo-Byte	Stringzeiger (1,0,..0)
----------	Wert Hi-Byte	Längen-Byte
Wert von (1,0,..0)	----------	Adresse Lo-Byte
Exponent	.	Adresse Hi-Byte
Mantisse MSB	.	----------
.	.	.
----------	----------	----------

```
------------        ------------        ------------
Wert von (N,N,..N)  Wert von (N,N,..N)  Stringzeiger (N,N,..N)
Exponent            Wert Lo-Byte        Längen-Byte
Mantisse MSB        Wert Hi-Byte        Adresse Lo-Byte
Mantisse            ============        Adresse Hi-Byte
Mantisse                                ============
Mantisse LSB
============
```

Wie wir sehen können, stehen numerische Werte direkt in den Tabellen. Bei den Zeichenketten sind dort nur Zeiger gespeichert.

6.2 Wer suchet, der findet

Applesoft besitzt eine Routine, die es uns ermöglicht, eine Variable im Speicher zu finden. PTRGET ($DFE3) kann sowohl für einfache als auch für Feldvariablen benutzt werden. Zuvor muß TXTPTR ($00B8/B9) auf den 1. Buchstaben des Variablennamens zeigen. Mit der Speicherstelle SUBFLAG ($0014) können wir noch festlegen, ob auch Ganzzahl-Variable (INTEGER) gesucht werden sollen. Ist Bit 7 von SUBFLAG gesetzt, sind Ganzzahl-Variable ausgeschlossen.

PTRGET hat eine Vielzahl von Ergebnissen. Gehen wir zunächst davon aus, daß die Variable bereits existiert. Dann zeigt VARPNT ($0083/84) auf die Speicheradresse des Variablen**wertes** (nicht auf den Namen!). Bei reellen Zahlen ist das der Exponent, bei ganzen Zahlen das Lo-Byte und bei Zeichenketten das Längenbyte. Das Lo-Byte dieser Adressse steht zusätzlich im Akkumulator, während sich das Hi-Byte im Y-Register befindet. LOWTR ($009B/9C) zeigt auf das 1. Namensbyte, während beide Bytes mit entsprechend gesetzten/gelöschten Bits 7 nach VARNAM ($0081/82) übertragen wurden. VALTYP ($0011) ist $00 bei einer numerischen Variablen und $FF bei einer Zeichenkette. NUMTYP ($0012) ist $00 bei einer reellen Zahl und $80 bei einer Ganzzahl. NUMDIM ($000F) ist $00 bei einer Feldvariablen, sonst <> $00. FLENGTH ($0064) beinhaltet bei Feldvariablen die Eintragslänge (2/5/3 für Ganzzahl/Reelle Zahl/Zeichenkette) und HIGHDS ($0094/95) zeigt – ebenfalls nur bei Feldvariablen – auf den ersten Eintrag im Werteblock der Tabelle. Existiert die Variable beim Aufruf von PTRGET noch nicht, wird sie angelegt

und auf Null initialisiert. Bei Feldvariablen geschieht dies mit einer Dimensionierung von 10 für jeden Index. War allerdings einer der Indizes beim Aufruf schon größer als 10, so erfolgt die Fehlermeldung „BAD SUBSCRIPT ERROR". Diese Fehlermeldung erscheint im übrigen auch, wenn eine Feldvariable von PTRGET gesucht wird, zu der vorher nur eine kleinere Matrix definiert wurde. Wenn Sie verhindern wollen, daß eine Variable neu angelegt wird, müssen Sie in SUBFLAG das Bit 6 setzen (z.B. $40). Vergessen Sie aber nicht, es später wieder zurückzusetzen, denn ein folgendes „INPUT" von Applesoft aus würde sonst nicht mehr funktionieren.

Speziell für Feldvariablen gibt es noch eine zweite Lokalisierroutine bei $F7D9 mit dem Namen GETARYPT. Auch hier muß TXTPTR auf das 1. Namensbyte zeigen. Ist die Variable nicht vorhanden, wird sie nicht angelegt, sondern eine Fehlermeldung ausgegeben. Wird die Variable gefunden, so zeigt LOWTR ($009B/9C) auf das 1. Byte des Namens im Strukturblock (Kopf).

PTRGET $DFE3
Sucht eine Variable (außer Funktionen!)

Eingabe:
TXTPTR ($00B8/B9) zeigt auf 1. Namensbyte
SUBFLAG ($0014) = $00, Bit 7 setzen, um Ganzzahlen zu sperren, Bit 6 setzen, um Neuanlage zu verhindern

Ausgabe:
Akku = Variablenwert Lo, Y-Reg = Variablenwert Hi
VARPNT ($0083/84) zeigt auf Variablenwert (Lo/Hi)
LOWTR ($009B/9C) zeigt auf 1. Namensbyte
VARNAM ($0081/82) enthält den Variablennamen
VALTYP ($0011) = $00 bei numerischen Var., = $FF bei Zeichenkette
NUMTYP ($0012) = $00 bei reeller Zahl, = $80 bei Ganzzahl
NUMDIM ($000F) = $00 bei Feldvariablen, sonst <> $00
FLENGTH ($0064) Eintragslänge bei Feldvariablen
HIGHDS ($0094/95) Zeiger auf Beginn des Werteblocks

GETARYPT $F7D9
sucht Variablenkopf einer Feldvariablen

Eingabe:
TXTPTR ($00B8/B9) zeigt auf 1. Namensbyte

Ausgabe:
LOWTR ($009B/9C) zeigt auf den Beginn des Strukturblocks (1. Namensbyte)

6.3 Hin und her, das fällt nicht schwer

Unser Wissen über die Applesoft-Variablen ist bisher recht theoretischer Natur. In einem kleinen Demonstrationsprogramm wollen wir jetzt üben, mit einem Maschinensprache-Programm Inhalte von Variablen zu füllen, die durch ein (gleichzeitig im Speicher befindliches) Applesoft-Programm angelegt wurden. Anschließend verarbeitet das BASIC-Programm die neuen Werte.

VAR-DEMO

```
10  PRINT  CHR$ (4)"BRUN VAR-DEMO.OBJ"
15  PRINT  CHR$ (21)
20  TEXT : HOME : VTAB 2: HTAB 8: PRINT "VARIABLEN DEMONSTRATION"
30  VTAB 5: PRINT "GEBEN SIE EINE ZAHL ZWISCHEN EINS UND"
    : PRINT "NEUN EIN: ";: GET Z$:Z =  VAL (Z$): IF Z < 1
    THEN  PRINT  CHR$ (7): GOTO 30
40  PRINT Z: POKE 254,Z
50  VTAB 8: PRINT "GEBEN SIE EIN BELIEBIGES DRUCKBARES":
    PRINT "ZEICHEN EIN: ";: GET Z$:Y =  ASC (Z$): IF Y < 32
    OR Y > 126 THEN  PRINT  CHR$ (7): GOTO 50
60  PRINT Z$: POKE 255,Y
70  Z$ = "----------------------------------------"
80  VTAB 11: PRINT Z$: PRINT : PRINT "JETZIGE VARIABLENWERTE:":
    PRINT Z$: PRINT : PRINT "A%= "A%;: HTAB 10: PRINT "A = "A;:
    HTAB 20: PRINT "A$= "A$
90  PRINT "C= "C: PRINT "B(A%)= "B(A%): PRINT "C$= "C$:
    PRINT : PRINT Z$
100  PRINT : PRINT "GEBEN SIE JETZT ZUNAECHST EINE
     FLIESS-  KOMMAZAHL IN DAS LAUFENDE MASCHINENPRO- GRAMM
     EIN (ENDE = <CR>) UND DANACH EINE ZEICHENKETTE (MAX.
     254 ZEICHEN):"
110  CALL 38156
```

```
120  PRINT : PRINT Z$: PRINT : PRINT "NEUE VARIABLENWERTE:":
     PRINT Z$: PRINT : PRINT "A%= "A%;: HTAB 10: PRINT "A = "A;:
     HTAB 20: PRINT "A$= "A$
125  PRINT "C= "C: PRINT "B(A%)= "B(A%): PRINT "C$= "C$:
     PRINT : PRINT Z$
130  PRINT : PRINT "ERNEUT? ";: GET Z$: IF Z$ = "J" OR
     Z$ = "j" OR Z$ = "Y" OR Z$ = "y" THEN  GOTO 20
140  END
```

Das aufgerufene Maschinenprogramm sieht so aus:

VARIABLEN-DEMO

```
                    1    ;*******************************
                    2    ;   AS-Variablen-Zuweisungen   *
                    3    ;   vom Maschinenprogramm aus  *
                    4    ;        Demo-Programm         *
                    5    ;  (C) 1985  Jürgen B. Kehrel  *
                    6    ;*******************************
                    7    ;
                    8    PROMPT    EQU $0033
                    9    LINNUM    EQU $0050
                   10    VARPNT    EQU $0083
                   11    FORPNT    EQU $0085
                   12    CHRGOT    EQU $00B7
                   13    TXTPTR    EQU $00B8
                   14    WERT      EQU $00FE
                   15    ZEICHEN   EQU $00FF
                   16    PUFFER    EQU $0200
                   17    GDBUFS    EQU $D539
                   18    STRCPY    EQU $DA9A
                   19    PTRGET    EQU $DFE3
                   20    SNGFLT    EQU $E301
                   21    STRLT1    EQU $E3E9
                   22    MOVEMF    EQU $EB2B
                   23    FIN       EQU $EC4A
                   24    RSHIMEM   EQU $F28C
                   25    GETLNZ    EQU $FD67
                   26    BASE      EQU $9500
                   27    ;
                   28              ORG BASE
                   29    ;
                   30    ; HIMEM: unter Programm setzen (DOS 3.3)
                   31    ;
9500: A9 00        32    START     LDA #<BASE
9502: 85 50        33              STA LINNUM
9504: A9 95        34              LDA #>BASE
9506: 85 51        35              STA LINNUM+1
9508: 20 8C F2     36              JSR RSHIMEM
950B: 60           37              RTS
                   38    ;
                   39    ; Textpointer von AS retten
                   40    ;
950C: A5 B8        41    CALL      LDA TXTPTR
950E: 8D B2 95     42              STA STXTPTR
9511: A5 B9        43              LDA TXTPTR+1
9513: 8D B3 95     44              STA STXTPTR+1
```

```
                        45  ;
                        46  ; Integer-Variable mit WERT füllen
                        47  ;
9516: A9 C8             48          LDA #<INTVAR    ;Textpointer auf
9518: 85 B8             49          STA TXTPTR      ;Variablennamen
951A: A9 95             50          LDA #>INTVAR    ;zeigen lassen
951C: 85 B9             51          STA TXTPTR+1
951E: 20 E3 DF          52          JSR PTRGET      ;Variable suchen
9521: A9 00             53          LDA #$00        ;Hi-Byte-Wert
9523: A8                54          TAY             ;Y=0
9524: 91 83             55          STA (VARPNT),Y  ;speichern
9526: C8                56          INY             ;auf Lo-Byte stellen
9527: A5 FE             57          LDA WERT        ;Lo-Byte-Wert laden
9529: 91 83             58          STA (VARPNT),Y  ;speichern
                        59  ;
                        60  ; Floating-Point-Variable mit WERT füllen
                        61  ;
952B: A4 FE             62          LDY WERT        ;WERT laden
952D: 20 01 E3          63          JSR SNGFLT      ;in Fließkomma-Zahl
9530: A9 CB             64          LDA #<FLOATVAR  ;Textpointer auf
9532: 85 B8             65          STA TXTPTR      ;Variablennamen
9534: A9 95             66          LDA #>FLOATVAR  ;zeigen lassen
9536: 85 B9             67          STA TXTPTR+1
9538: 20 E3 DF          68          JSR PTRGET      ;Variable suchen
953B: AA                69          TAX             ;Lo-Byte Adr. nach X
953C: 20 2B EB          70          JSR MOVEMF      ;FAC -> Variable
                        71  ;
                        72  ; FP-Variable mit eingegebenem Wert füllen
                        73  ;
953F: A9 A1             74          LDA #"!"        ;Prompt ändern
9541: 85 33             75          STA PROMPT
9543: 20 67 FD          76          JSR GETLNZ      ;Zeile holen
9546: 20 39 D5          77          JSR GDBUFS      ;Bit7 löschen, Endmarker
9549: A9 00             78          LDA #<PUFFER    ;Textpointer auf
954B: 85 B8             79          STA TXTPTR      ;Eingabe zeigen
954D: A9 02             80          LDA #>PUFFER    ;lassen
954F: 85 B9             81          STA TXTPTR+1
9551: 20 B7 00          82          JSR CHRGOT      ;1 Zeichen im Akku
9554: 20 4A EC          83          JSR FIN         ;Puffer -> FAC
9557: A9 D9             84          LDA #<FPEING    ;Textpointer auf
9559: 85 B8             85          STA TXTPTR      ;Variablennamen
955B: A9 95             86          LDA #>FPEING    ;zeigen lassen
955D: 85 B9             87          STA TXTPTR+1
955F: 20 E3 DF          88          JSR PTRGET      ;Variable suchen
9562: AA                89          TAX             ;X vorbereiten
9563: 20 2B EB          90          JSR MOVEMF      ;FAC -> Variable
                        91  ;
                        92  ; 1 Zeichen einem String zuweisen
                        93  ;
9566: A5 FF             94          LDA ZEICHEN     ;ZEICHEN laden
9568: 8D 00 02          95          STA PUFFER      ;in Eingabepuffer
956B: A2 01             96          LDX #$01        ;Stringlänge = 1
956D: 20 39 D5          97          JSR GDBUFS      ;Bit7 löschen, "0" Ende
9570: A9 CD             98          LDA #<STRVAR    ;Textpointer auf
```

```
9572: 85 B8      99             STA TXTPTR     ;Variablennamen
9574: A9 95      100            LDA #>STRVAR   ;zeigen lassen
9576: 85 B9      101            STA TXTPTR+1
9578: 20 B4 95   102            JSR MAKESTR    ;String herstellen
                 103 ;
                 104 ; Arrayindex auch eine Variable
                 105 ;
957B: A9 D0      106            LDA #<ARVAR    ;zuerst Textpointer
957D: 85 B8      107            STA TXTPTR     ;auf Variablennamen
957F: A9 95      108            LDA #>ARVAR    ;zeigen lassen
9581: 85 B9      109            STA TXTPTR+1
9583: 20 E3 DF   110            JSR PTRGET     ;Variable suchen
9586: A4 FE      111            LDY WERT       ;WERT laden
9588: 20 01 E3   112            JSR SNGFLT     ;WERT -> FAC
958B: A6 83      113            LDX VARPNT     ;Variablenzeiger
958D: A4 84      114            LDY VARPNT+1   ;zurückladen
958F: 20 2B EB   115            JSR MOVEMF     ;FAC -> Variable
                 116 ;
                 117 ; Stringeingabe speichern
                 118 ;
9592: A9 BE      119            LDA #">"       ;Prompt-Zeichen ändern
9594: 85 33      120            STA PROMPT
9596: 20 67 FD   121            JSR GETLNZ     ;Zeile holen
9599: 20 39 D5   122            JSR GDBUFS     ;Bit7 löschen, "0" Ende
959C: A9 D6      123            LDA #<EINGVAR  ;Textpointer auf
959E: 85 B8      124            STA TXTPTR     ;Variablennamen
95A0: A9 95      125            LDA #>EINGVAR  ;zeigen lassen
95A2: 85 B9      126            STA TXTPTR+1
95A4: 20 B4 95   127            JSR MAKESTR    ;String herstellen
                 128 ;
                 129 ; alten Textpointer wieder herstellen
                 130 ;
95A7: AD B2 95   131            LDA STXTPTR
95AA: 85 B8      132            STA TXTPTR
95AC: AD B3 95   133            LDA STXTPTR+1
95AF: 85 B9      134            STA TXTPTR+1
95B1: 60         135            RTS
                 136 ;
                 137 STXTPTR     DFS 2
                 138 ;
                 139 ; Zeichenkette im Eingabepuffer
                 140 ; in den bezeichneten String kopieren
                 141 ;
95B4: 20 E3 DF   142 MAKESTR    JSR PTRGET     ;Variable suchen
95B7: 85 85      143            STA FORPNT     ;Zeiger speichern
95B9: 84 86      144            STY FORPNT+1
95BB: A9 00      145            LDA #<PUFFER   ;Lo Stringadresse
95BD: A0 02      146            LDY #>PUFFER   ;Hi Stringadresse
95BF: A2 00      147            LDX #$00       ;Endzeichen
95C1: 20 E9 E3   148            JSR STRLT1     ;String kopieren
95C4: 20 9A DA   149            JSR STRCPY     ;Descriptor -> Var.
95C7: 60         150            RTS            ;fertig
                 151 ;
                 152 ; Variablennamen, Ende mit ":"
                 153 ;
```

```
95C8: 41 25 3A  154  INTVAR    ASC 'A%:'
95CB: 41 3A     155  FLOATVAR  ASC 'A:'
95CD: 41 24 3A  156  STRVAR    ASC 'A$:'
95D0: 42 28 41  157  ARVAR     ASC 'B(A%):'
95D6: 43 24 3A  158  EINGVAR   ASC 'C$:'
95D9: 43 3A     159  FPEING    ASC 'C:'
```

Das BASIC-Programm startet zunächst das Maschinenprogramm, das seinerseits HIMEM unter sich legt, um sich vor einem Überschreiben durch die BASIC-Variablen zu schützen (nur unter DOS 3.3!). Sodann liest das BASIC-Programm eine Zahl zwischen 1 und 9 und ein druckbares Zeichen ein. Die Zahl und der ASCII-Wert des Zeichens werden in die Speicherstellen 254 und 255 ($00FE und $00FF) gepoked, um sie dort zwischenzuspeichern.
Im Anschluß daran werden alle Variablen mit ihrem augenblicklichen Inhalt angezeigt, damit Sie sehen können, daß Ihre Eingaben dort noch nicht angekommen sind. Nach der Ausgabe einer entsprechenden Aufforderung wird mit dem CALL in die Speicherstelle $950C gesprungen. Nach dem Ablauf des Maschinenprogramms zeigt das BASIC-Programm erneut die Variablen-Werte, um Ihnen zu beweisen, daß jetzt die neuen Inhalte angekommen sind. Sie können die Demo immer wiederholen.
Der für uns jetzt interessantere Teil liegt im Assembler-Programm. Dieses rettet zunächst den Applesoft-Textzeiger TXTPTR, um ihn später wieder herstellen zu können. Würde dies nicht geschehen, könnte das BASIC-Programm abstürzen. Da die Variablen mit PTRGET gesucht werden sollen, muß nämlich TXTPTR auf den Variablennamen umgesetzt werden. Die Variablennamen stehen am Programmende in einer Tabelle. Jeder Name muß mit dem Trennzeichen „:" beendet werden, damit PTRGET die richtige Namenslänge erkennt. In Ihren eigenen Programmen müssen die Namen nicht fest vorgegeben sein. Es genügt, sie an eine definierte Stelle zu schreiben, was auch in einem laufenden Programm geschehen kann.
Zuerst soll die Ganzzahl-Variable „A%" gefüllt werden. Wir stellen deshalb TXTPTR auf den Namensanfang und rufen PTRGET auf. VARPNT zeigt anschließend auf den Variablenwert innerhalb der Tabelle. Als Zahlenwert wollen wir die Zahl nehmen, die vom Applesoftprogramm in der Speicherstelle $00FE (254) abgelegt worden ist. Da diese nur 1 Byte benötigt, muß das Hi-Byte der Variablen auf Null gesetzt werden. Durch die indirekte indizierte Adressierung werden die beiden Wert-Bytes direkt in die Variable geschrieben (Zeile 53 – 58).
Dieselbe Zahl soll anschließend als Fließkomma-Zahl in „A" untergebracht werden. Mit SNGFLT wird sie in eine ungepackte Fließkomma-Zahl im MFAC

verwandelt. Der Textpointer wird auf den neuen Namen umgesetzt und PTRGET findet wieder den richtigen Speicherplatz. Mittels MOVEMF wird die Fließkomma-Zahl gepackt und dann auf den Speicherplatz der Variablen transportiert. MOVEMF erwartet das Lo-Byte der Zieladresse im X-Register und das Hi-Byte im Y-Register. PTRGET hat dies für das Hi-Byte auch vorbereitet, den Lo-Wert aber im Akkumulator übergeben. Ein TAX (Zeile 89) sorgt hier für die Anpassung.

Eine über die Tastatur eingegebene Zahl soll in „C" gepeichert werden. Wir benutzen die Routine GETLNZ ($FD67), um alle Zeichen von der Tastatur bis zum abschließenden RETURN zu speichern. GETLNZ gibt zunächst ein $8D (= RETURN) über COUT aus und danach das Zeichen, das in PROMPT ($0033) enthalten ist. Wir haben PROMPT in diesem Fall vorher mit einem Ausrufezeichen gefüllt. Alle Zeichen von der Tastatur (max. 255) werden nacheinander ab $0200 mit gesetztem Bit 7 abgelegt, wobei das X-Register die Anzahl mitzählt.

GETLNZ $FD67
Gibt ein RETURN und den Inhalt von PROMPT über COUT aus. Eine Eingabe wird über den Vektor KSWL/H ($0038/39) bis zu einem abschließenden RETURN geholt und in den Tastaturpuffer ab $200 mit gesetztem Bit 7 abgelegt.

Eingabe: PROMPT ($0033) = $A0 – $FE

Ausgabe:
Ab $0200 Eingabestring
X-Reg = Eingabelänge (zeigt auf $8D)

GETLN $FD6A
Wie GETLNZ, nur wird kein RETURN zu Beginn ausgegeben.

Eingabe: PROMPT ($0033) = $A0 – $FE

Ausgabe:
Ab $0200 Eingabestring
X-Reg = Eingabelänge (zeigt auf $8D)

GETLN1 $FD6F
Wie GETLNZ, nur wird weder RETURN noch PROMPT zu Beginn ausgegeben.

Eingabe: –

Ausgabe:
Ab $0200 Eingabestring
X-Reg = Eingabelänge (zeigt auf $8D)

Die Applesoft-Routine GDBUFS ($D539) löscht bei der im X-Register übergebenen Anzahl von Bytes im Tastaturpuffer die Bits 7 und hängt ein $00 als Endkennzeichnung der Eingabe an. In diesem Fall wird das $8D durch $00 überschrieben.

GDBUFS $D539
Fügt $00 als Endmarke an die Eingabe im Tastaturpuffer und löscht in allen Bytes das Bit 7.

Eingabe: X-Reg = Anzahl der Bytes

Ausgabe: Akku = $00, X-Reg = $FF, Y-Reg = $01

Die Zeichenkette im Puffer, die die gewünschte Zahl darstellt, ist noch ASCII-kodiert. Wir müssen sie in eine Fließkomma-Zahl umwandeln. Der Applesoft-Interpreter stellt uns dafür FIN ($EC4A) zur Verfügung, das einen Zahlenstring in eine FP-Konstante im MFAC umwandelt. Vor dem Aufruf von FIN muß TXTPTR auf das erste Stringzeichen weisen, und mit einem JSR CHRGOT ($00B7) muß dieses 1. Zeichen in den Akkumulator geholt werden.

FIN $EC4A
Verwandelt Zahlenstring in FP-Konstante im MFAC

Eingabe:
TXTPTR ($00B8/B9) zeigt auf 1. Stringzeichen
Akku = 1. Stringzeichen

Ausgabe:
FP-Konstante im MFAC
TXTPTR zeigt hinter Zahlenstring

Jetzt wird der TXTPTR auf den Variablennamen umgesetzt und mit PTRGET in bekannter Manier die Variable gesucht. MOVEMF transportiert dann die Fließkommazahl in gepackter Form vom MFAC in die Variable.

Unsere nächste Aufgabe besteht darin, ein ASCII-Zeichen aus der Speicherstelle $FF (255) in die Variable „A$" zu bringen. Es wird dazu aus seiner Speicherstelle in den Tastatureingabe-Puffer nach $0200 kopiert. GDBUFS löscht Bit 7 und hängt ein $00 an.
Der Textpointer TXTPTR wird wieder auf den Variablennamen gesetzt und die Unterroutine MAKESTR aufgerufen, die die Zeichenkette im Tastatur-Puffer bis zum abschließenden $00 in den bezeichneten String kopiert. Auch hier setzen wir PTRGET ein, nur daß wir die Zeiger auf die Variable zunächst nach FORPNT und FORPNT+1 zwischenspeichern. Die Routine STRLT1 ($E3E9) bestimmt die Länge des Strings anhand des im X-Register übergebenen Endmarkers. Wenn der String auf der Zero-Page oder im Tastatur-Puffer steht, wird er in den Stringpool unterhalb von FRETOP kopiert und FRETOP nach unten gesetzt. Ein Deskriptor für den String wird angelegt und auf den sogenannten „Deskriptoren-Stack" zur Zwischenspeicherung geschoben. TEMPPT ($0052) ist der Zeiger („Stackpointer") im Deskriptorenstack, der entsprechend korrigiert wird. Wenn wir aus einem BASIC-Programm kommen, brauchen wir ihn nicht zu initialisieren.

STRLT1 $E3E9
Bestimmt Stringlänge anhand des Endmarkers. Wenn der String in Seite 0 oder 2 liegt, wird er in den Stringpool kopiert. Ein Deskpritor wird auf dem Deskriptoren-Stack angelegt.

Eingabe:
X-Reg = Endmarker, Akku = Lo-Adresse String
Y-Reg = Hi-Adresse String, TEMPPT = Deskr.-Stack-Zeiger

Ausgabe:
TEMPPT um 3 heraufgesetzt (= X-Reg)
FRETOP gegebenenfalls versetzt
VALTYP ($0011) = $FF bei String
VPNT ($00A0/A1) zeigt auf Deskr.-Stack
STRNG2 ($00AD/AE) zeigt hinter den String

Vom Deskriptoren-Stack müssen die drei Bytes des String-Deskriptors noch in die Variable gebracht werden. STRCPY ($DA9A) besorgt dies für uns, wenn FORPNT ($0085/86) auf die Variable und VPNT ($00A0/A1) auf den Deskriptor zeigt. VPNT wurde durch STRLT1 richtig gesetzt, und FORPNT erhielt seine Werte bereits in Zeile 143/44.

STRCPY $DA9A
Überträgt Deskriptor in die Variable

Eingabe:
FORPNT ($0085/86) zeigt auf Variable
VPNT ($00A0/A1) zeigt auf Deskriptor

Ausgabe: –

Unser nächstes Ziel ist es, eine Feldvariable zu füllen, deren Index auch wieder eine Variable ist. Die Adresse der Variablen wird in der bekannten Weise über PTRGET gesucht. Anschließend wird ihr Inhalt aus der Speicherstelle $00FE (254) geladen. Da wir als Typ eine reelle Variable gewählt haben, muß der Wert in eine Fließkomma-Zahl umgewandelt werden. SNGFLT „floatet" das Y-Regi-

ster nach MFAC und MOVEMF besorgt erneut den Transport in die Variable. X- und Y-Register müssen hierbei neu mit der Zieladresse geladen werden, da die beiden voranstehenden Befehle die von PTRGET im Akku und im Y-Register übergebenen Werte zerstört haben. VARPNT, das dieselbe Information enthält, ist aber erhalten geblieben.
Einen beliebigen String von der Tastatur zu lesen, fällt nun nicht mehr schwer. GETLNZ nimmt wieder unsere Eingabe entgegen, GDBUFS löscht Bit 7 und setzt den Endmarker $00, während die Unterroutine MAKESTR in der schon beschriebenen Art und Weise die Zeichenkette aus dem Tastatur-Puffer in den bezeichneten String kopiert.
Zum Abschluß der Demonstration wird TXTPTR auf seinen alten Wert zurückgesetzt und nach BASIC zurückgekehrt.

In diesem Demonstrationsprogramm sind fast alle gebräuchlichen Fälle vorgekommen, so daß Sie für Ihre Programme die richtigen Teile nur herauszupicken brauchen. Eines der schwierigsten Probleme im Zusammenspiel von Assembler und Applesoft ist damit auf einfache Weise zu lösen.

6.4 Stühlerücken

Weil wir jetzt wissen, wie Variablen des Applesoft aufgebaut sind und wie wir sie im Speicher lokalisieren können, schreiben wir uns nun ein paar nützliche Anwendungen.
Wenn Sie in einem BASIC-Programm zwei Variablen miteinander tauschen wollen (z.B. A$ und B$), müssen Sie immer einen Ringtausch programmieren:

```
T$ = A$ : A$ = B$ : B$ = T$
```

Das geht relativ langsam, was aber zumeist nicht sehr störend ist. Gravierender ist die Tatsache, daß ein solcher Ringtausch viele sogenannte „Stringleichen" erzeugt, da Applesoft für jede Neuzuweisung an einen bestehenden Variablennamen einen neuen Eintrag im Stringpool macht. Damit füllt sich der Speicher und Ihr Programm legt eine Zwangspause ein, um alles wieder aufzuräumen. Viel eleganter ist es da doch, in einem Maschinenprogramm die beiden Variablen zu suchen und direkt ihre Eintragungen zu vertauschen. Wir wollen dieses Programm SWAP nennen. Damit wir es auch von Applesoft aufrufen können, müssen wir uns noch eine bequeme Übergabe ausdenken.

Die Programmierer von Applesoft-BASIC haben in den Interpreter mit dem &-Befehl (= Ampersand) bereits eine Möglichkeit eingebaut, zusätzliche Maschinenroutinen aufzurufen. Immer wenn der Interpreter auf ein nicht in Anführungszeichen oder nach einem REM stehenden &-Zeichen im Programm trifft, macht er einen Sprung nach $03F5. Dort steht normalerweise ein JMP $FF58. Bei $FF58 im Monitor steht ein RTS, das uns wieder ins Applesoft-Programm zurückbefördert. Wenn wir alles so beließen, hätte dieser Sprung keine Auswirkungen. Wir können aber das JMP $FF58, da es im RAM steht, auch verändern und in ein von uns geschriebenes Maschinenprogramm umlenken. Von dort kehren wir dann nach getaner Arbeit mit einem RTS zurück ins laufende Applesoft-Programm. Dieser „Trick" ist ausreichend, wenn wir keine weitere Information zwischen BASIC und dem Maschinenprogramm zu übergeben haben. Unserer Routine SWAP müssen wir aber die Namen der zwei zu vertauschenden Variablen nennen.
Um die notwendige Parameterübergabe zu verstehen, müssen wir zunächst erklären, wie Applesoft das BASIC-Programm liest. Beim Kaltstart des Rechners wird aus dem ROM von $F10B nach $00B1 bis $00C8 ein kleines Programm kopiert, die CHRGET-Routine ($00B1), zu der auch der Textpointer TXTPTR gehört. CHRGET hat zwei Einsprünge: bei $00B1 wird zunächst der TXTPTR um 1 erhöht und dann der Akkumulator mit dem Byte von der Textpointer-Stelle geladen. Bei $00B7 (= CHRGOT) wird der Akkumulator geladen, ohne daß vorher der Textpointer weitergesetzt wurde.

CHRGET $00B1
Erhöht TXTPTR und läd TXTPTR-Stelle in den Akku. Leerzeichen ($20) werden übersprungen. Testet Zeichen.

Eingabe: –

Ausgabe: Akku = gelesenes Zeichen
Carry-Bit = 1 ⇒ Ziffer
Carry-Bit = 0 ⇒ sonst. Zeichen
Zero-Bit = 1 ⇒ " oder $00
Zero-Bit = 0 ⇒ weder " noch $00

CHRGOT $00B7
Läd Akku von der TXTPTR-Stelle. Leerzeichen ($20) werden übersprungen. Testet Zeichen.

Eingabe: –

Ausgabe: Akku = gelesenes Zeichen
Carry-Bit = 1 ⇒ Ziffer
Carry-Bit = 0 ⇒ sonst. Zeichen
Zero-Bit = 1 ⇒ " oder $00
Zero-Bit = 0 ⇒ weder " noch $00

Wenn nun Applesoft auf ein & trifft, springt es nicht nur nach $03F5, sondern führt auch noch ein JSR CHRGET aus, so daß sich das auf & folgende Zeichen im Akkumulator befindet und TXTPTR **hinter** das & zeigt. Mit diesem Wissen können wir jetzt eine Routine schreiben, die alle nach dem & stehenden Zeichen als Parameter auswertet.

SWAP1

```
                      1   ;*******************************
                      2   ;   Variablen-Tauscher SWAP1   *
                      3   ;*******************************
                      4   ;
                      5   VARNAM    EQU $0081       ;+ $0082
                      6   VARPNT    EQU $0083       ;+ $0084
                      7   FORPNT    EQU $0085       ;+ $0086
                      8   AMPER     EQU $03F5
                      9   DATA      EQU $D995
                      10  MISMTCH   EQU $DD76
                      11  CHKCOM    EQU $DEBE
                      12  PTRGET    EQU $DFE3
                      13  ;
                      14            ORG $0300
                      15  ;
0300: A9 4C           16            LDA #$4C        ;JMP
0302: 8D F5 03        17            STA AMPER
0305: A9 10           18            LDA #<START
0307: 8D F6 03        19            STA AMPER+1     ;Ampersand Sprung-
030A: A9 03           20            LDA #>START     ;vektor setzen
030C: 8D F7 03        21            STA AMPER+2
030F: 60              22            RTS
                      23  ;
0310: 20 E3 DF        24  START     JSR PTRGET      ;1. Variable
0313: 85 85           25            STA FORPNT      ;Lo
0315: 84 86           26            STY FORPNT+1    ;Hi Adresse
0317: A5 81           27            LDA VARNAM      ;1. Namenszeichen
0319: 48              28            PHA             ;retten
```

```
031A: A5 82     29              LDA VARNAM+1    ;2. Namenszeichen
031C: 48        30              PHA             ;retten
031D: 20 BE DE  31              JSR CHKCOM      ; ,
0320: 20 E3 DF  32              JSR PTRGET      ;2. Variable
0323: 68        33              PLA             ;alter Name
0324: 45 82     34              EOR VARNAM+1    ;gleicher Typ?
0326: 30 17     35              BMI MISMATCH
0328: 68        36              PLA
0329: 45 81     37              EOR VARNAM      ;dito
032B: 30 12     38              BMI MISMATCH    ;geht nicht
032D: A0 04     39              LDY #$04        ;5 Bytes
032F: B1 83     40   LOOP       LDA (VARPNT),Y
0331: AA        41              TAX             ;retten
0332: B1 85     42              LDA (FORPNT),Y
0334: 91 83     43              STA (VARPNT),Y ;1. Hälfte
0336: 8A        44              TXA
0337: 91 85     45              STA (FORPNT),Y ;2. Hälfte
0339: 88        46              DEY
033A: 10 F3     47              BPL LOOP
033C: 4C 95 D9  48              JMP DATA        ;zum nächsten Befehl
                49   ;
033F: 4C 76 DD  50   MISMATCH   JMP MISMTCH     ;Fehlermeldung
```

Die Zeilen 16 bis 22 stellen ein Startprogramm dar, das den Ampersand-Vektor auf den Beginn des eigentlichen Programms (START) umsetzt. Dies muß nur ein einziges Mal geschehen, z.B. indem SWAP1 mittels BRUN gestartet wird. Bei jedem &-Zeichen springt dann der Interpreter nach START. Für den Aufruf vereinbaren wir folgende Syntax:

& 1.Variable, 2.Variable.

Wie wir wissen, zeigt TXTPTR auf das erste Zeichen nach dem &, hier also auf den Beginn des ersten Variablennamens. Wir können deshalb direkt PTRGET aufrufen, um die Variable zu finden. Die gefundene Adresse speichern wir wieder nach FORPNT zwischen, während der Variablenname (die beiden signifikanten Zeichen) auf den Stack gerettet wird (Z. 25 – 30). Auch PTRGET setzt den TXTPTR weiter, der nun auf das Komma zeigt. CHKCOM ($DEBE) prüft das Vorhandensein des Kommas und gibt im Fehlerfall ein „SYNTAX ERROR“ aus. Mit PTRGET suchen wir dann auch die zweite Variable.

Da wir nur Variablen gleichen Typs vertauschen können, wird die Übereinstimmung anhand der Bits 7 geprüft. Wenn das erste und das zweite Namensbyte jeweils paarweise miteinander EOR-verknüpft werden, muß das Negativ-Bit im Statusregister gelöscht werden, da sich bei Gleichheit nie eine 1 ergeben kann. Bei Nichtübereinstimmung wird ein „TYPE MISMATCH ERROR“ ausgegeben, den wir direkt im ROM aufrufen.

CHKCOM $DEBE
Prüft, ob TXTPTR auf ein Komma ($2C) zeigt

Eingabe: TXTPTR zeigt auf zu prüfendes Zeichen

Ausgabe:
Falls ja: führt CHRGET aus
Falls nein: Fehlermeldung „SYNTAX ERROR“

CHKOPN $DEBB
Prüft, ob TXTPTR auf eine geöffnete Klammer ($28) zeigt

Eingabe: TXTPTR zeigt auf zu prüfendes Zeichen

Ausgabe:
Falls ja: führt CHRGET aus
Falls nein: Fehlermeldung „SYNTAX ERROR“

CHKCLS $DEB8
Prüft, ob TXTPTR auf eine geschlossene Klammer ($29) zeigt

Eingabe: TXTPTR zeigt auf zu prüfendes Zeichen

Ausgabe:
Falls ja: führt CHRGET aus
Falls nein: Fehlermeldung „SYNTAX ERROR“

SYNCHR $DEC0
Prüft, ob das Zeichen unter dem TXTPTR und der Akkumulator übereinstimmen.

Eingabe:
TXTPTR zeigt auf zu prüfendes Zeichen
Akku = Vergleichszeichen

Ausgabe:
Falls ja: führt CHRGET aus
Falls nein: Fehlermeldung „SYNTAX ERROR"

Um die Inhalte unserer zwei Variablen zu tauschen, werden in einer Schleife (Zeile 39 – 47) die 5 Bytes des Variablenwertes vertauscht, wobei uns das X-Register als Zwischenspeicher dient. Unser Programm endet mit einem Sprung nach DATA ($D995), das den TXTPTR bis zum nächsten „:" oder bis zum Ende der BASIC-Zeile vorschiebt und dann mit RTS endet. Auf diese Weise gelangen wir in unser aufrufendes BASIC-Programm zurück.

DATA $D995
Schiebt TXTPTR bis zum Befehlsende ($3A oder $00) weiter

Eingabe: –

Ausgabe: TXTPTR auf nächstes $3A oder $00 gesetzt

Damit Sie SWAP auch gleich benutzen können, folgt ein kurzes Demo-Programm.

SWAP1.DEMO

```
1   REM  **** SWAP1 - DEMO ****
5   HOME
10  PRINT  CHR$ (4)"BRUN SWAP1.OBJ"
20  A$ = "EINS":B$ = "ZWEI"
30  PRINT "A$="A$" B$="B$: GOSUB 130
40  & A$,B$
50  PRINT "A$="A$" B$="B$
60  A = 123.45:B = 678.91
70  PRINT : PRINT "A="A" B="B: GOSUB 130
80  & A,B
```

```
90  PRINT "A="A" B="B
100  PRINT : PRINT : LIST 110,120
110  REM  JETZT FEHLER:
120  & A$,B
130  INVERSE : PRINT "  SWAP  ": NORMAL : RETURN
```

Wie Sie leicht ausprobieren können, funktioniert der Tausch sowohl der beiden Zeichenketten als auch der Fließkomma-Zahlen. Erst in Zeile 120 erhalten wir eine Fehlermeldung, da beide Variablen nicht zum selben Typ gehören. Nebenbei bemerkt: das BASIC-Programm hat noch einen weiteren Fehler, da der Ablauf in die Unterroutine 130 führen und dort einen „Return without Gosub"-Fehler bewirken würde. Dazu kommt es allerdings nicht, da bereits der Fehler in Zeile 120 das Programm zwangsweise beendet.

Unsere Swap-Routine ist schon ganz praktisch, allerdings versagt sie bei Feldvariablen. Wie Sie aus Kapitel 6.1 wissen, sind bei Feldvariablen die Einträge unterschiedlich lang, sodaß unser Austausch von 5 Bytes nur für reelle Zahlen funktionieren, dagegen bei Zeichenketten und Ganzzahlen Unordnung stiften würde. Mit einer kleinen Programmänderung können wir aber auch diesen Mangel noch beheben.

SWAP2

```
                        1   ;*******************************
                        2   ;   Variablen-Tauscher SWAP2   *
                        3   ;*******************************
                        4   ;
                        5   FLENGTH   EQU $0064
                        6   VARNAM    EQU $0081       ;+ $0082
                        7   VARPNT    EQU $0083       ;+ $0084
                        8   FORPNT    EQU $0085       ;+ $0086
                        9   AMPER     EQU $03F5
                        10  DATA      EQU $D995
                        11  MISMTCH   EQU $DD76
                        12  CHKCOM    EQU $DEBE
                        13  PTRGET    EQU $DFE3
                        14  ;
                        15            ORG $0300
                        16  ;
0300: A9 4C             17            LDA #$4C        ;JMP
0302: 8D F5 03          18            STA AMPER
0305: A9 10             19            LDA #<START
0307: 8D F6 03          20            STA AMPER+1     ;Ampersand Sprung-
030A: A9 03             21            LDA #>START     ;vektor setzen
030C: 8D F7 03          22            STA AMPER+2
030F: 60                23            RTS
                        24  ;
0310: 20 E3 DF          25  START     JSR PTRGET      ;1. Variable
0313: 85 85             26            STA FORPNT      ;Lo
0315: 84 86             27            STY FORPNT+1    ;Hi Adresse
0317: A5 81             28            LDA VARNAM      ;1. Namenszeichen
```

```
0319: 48          29              PHA             ;retten
031A: A5 82       30              LDA VARNAM+1    ;2. Namenszeichen
031C: 48          31              PHA             ;retten
031D: 20 BE DE    32              JSR CHKCOM      ; ,
0320: 20 E3 DF    33              JSR PTRGET      ;2. Variable
0323: 68          34              PLA             ;alter Name
0324: 45 82       35              EOR VARNAM+1    ;gleicher Typ?
0326: 30 18       36              BMI MISMATCH
0328: 68          37              PLA
0329: 45 81       38              EOR VARNAM      ;dito
032B: 30 13       39              BMI MISMATCH    ;geht nicht
032D: A4 64       40              LDY FLENGTH     ;2/3/5:INT/STR/REAL
032F: 88          41              DEY
0330: B1 83       42   LOOP       LDA (VARPNT),Y
0332: AA          43              TAX             ;retten
0333: B1 85       44              LDA (FORPNT),Y
0335: 91 83       45              STA (VARPNT),Y ;1. Hälfte
0337: 8A          46              TXA
0338: 91 85       47              STA (FORPNT),Y ;2. Hälfte
033A: 88          48              DEY
033B: 10 F3       49              BPL LOOP
033D: 4C 95 D9    50              JMP DATA        ;zum nächsten Befehl
                  51   ;
0340: 4C 76 DD    52   MISMATCH   JMP MISMTCH     ;Fehlermeldung
```

Die Modifikation ist nur gering, aber sehr wirkungsvoll. PTRGET legt bei Feldvariablen (nur dort!!) in der Speicherstelle FLENGTH ($0064) die Länge des Eintrags ab. Wir benutzen diesen Wert, um unsere Schleife zu initialisieren. Auch hier soll Ihnen wieder eine Demo zeigen, daß unser Programm erfolgreich arbeitet.

SWAP2.DEMO

```
1   REM  **** SWAP2 - DEMO ****
5   PRINT  CHR$ (4)"BRUN SWAP2.OBJ"
10  HOME : DIM A(2)
20  A(1) = 10:A(2) = 20: PRINT A(1)"    "A(2)
30  & A(1),A(2)
40  PRINT : INVERSE : PRINT "  SWAP  ": NORMAL
50  PRINT : PRINT A(1)"    "A(2)
60  END
```

Achtung: SWAP1 nur für einfache Variablen, SWAP2 nur für Feldvariablen benutzen!

6.5 Etwas Ordnung kann nicht schaden

Mit der Routine SWAP haben wir schon einen wichtigen Teil für ein Sortierprogramm geschrieben. Wir wollen hier keine allzu komplizierten Algorithmen entwickeln, sondern ein einfaches und kurzes Programm benutzen, das auf der Seite 3 Platz hat und einfach indizierte Zeichenketten-Felder aufsteigend nach ihren ASCII-Werten sortiert. Wir verwenden einen Blasen- oder „Bubble"-Sortieralgorithmus, der zwar nicht übermäßig schnell, dafür aber einfach zu verstehen ist.

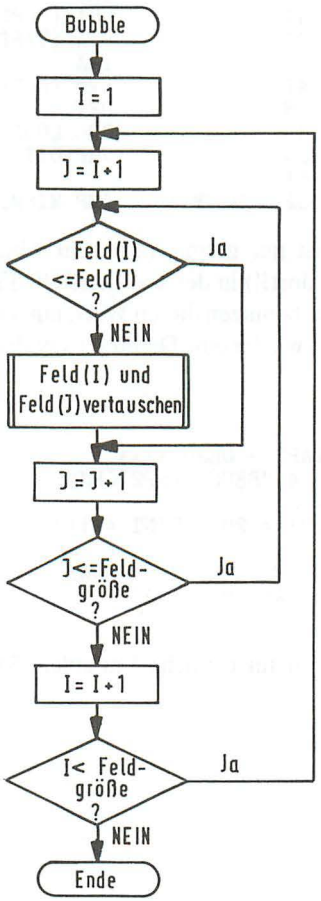


Abb. 9: „Bubble"-Sort

Ein BASIC-Programm könnte folgendermaßen aussehen:

BASIC.SORT

```
1   REM  *** SORTIER-DEMONSTRATION ***
10  CLEAR : TEXT
20  HOME : VTAB 5: INPUT "ANZAHL DER ZU SORTIERENDEN STRINGS? ";N
30  DIM A$(N)
40  B$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
50  HOME : VTAB 5: FLASH : PRINT "STRINGS WERDEN ERZEUGT": NORMAL
60  FOR I = 1 TO N: FOR J = 1 TO 10
70  A$(I) = A$(I) +  MID$ (B$, INT ( RND (1) * 26) + 1,1)
80  NEXT J,I
90  HOME : FOR I = 1 TO N: PRINT A$(I): NEXT : PRINT
100  INPUT "‹RETURN› SORTIERBEGINN";C$
110  M = N: REM  CA. 77 SEKUNDEN BEI N=100
120  FOR I = 1 TO M - 1
125  PRINT ".";
130  FOR J = I + 1 TO M
140  IF A$(I) ‹  = A$(J) THEN 160
150  H$ = A$(I):A$(I) = A$(J):A$(J) = H$
160  NEXT J,I
170  PRINT : INPUT "‹RETURN› FUER SORTIERTE LISTE";C$
180  HOME : VTAB 5: FOR I = 1 TO N
190  PRINT I;: POKE 36,5: PRINT A$(I): NEXT I
200  END
```

Damit Sie den Verlauf besser verfolgen können, wird bei jedem Durchgang durch die äußere Schleife ein Punkt ausgegeben. Für 100 Zeichenketten mit der konstanten Länge 10 benötigt das BASIC-Programm ca. 77 Sekunden. Das folgende Maschinenprogramm verkürzt diese Zeit auf ca. 3,5 Sekunden. Es ist also gut 20-mal schneller.

BSORT

```
1     ;*********************************
2     ;   „Bubble-Sortierer" BSORT    *
3     ;*********************************
4     ;
5     ZAHL      EQU $0000       ;+ $0001
6     COUNT     EQU $0002       ;+ $0003
7     LINKS     EQU $0004       ;+ $0005
8     RECHTS    EQU $0006       ;+ $0007
9     SUBFLAG   EQU $0014
10    CMPFLG    EQU $0016
11    VARNAM    EQU $0081       ;+ $0082
12    LOWTR     EQU $009B       ;+ $009C
13    MFAC      EQU $009D
14    SFAC      EQU $00A5
15    AMPER     EQU $03F5
16    DATA      EQU $D995
17    MISMTCH   EQU $DD76
18    STRCMP    EQU $DF7D
19    PTRGET    EQU $DFE3
20    BSSERR    EQU $E196
21    ;
22              ORG $0300
23    ;
```

```
                        24  ; &-Vektor setzen
                        25  ;
0300: A9 4C             26          LDA #$4C        ;JMP
0302: 8D F5 03          27          STA AMPER
0305: A9 10             28          LDA #<START
0307: 8D F6 03          29          STA AMPER+1     ;Ampersand Sprung-
030A: A9 03             30          LDA #>START     ;vektor setzen
030C: 8D F7 03          31          STA AMPER+2
030F: 60                32          RTS
                        33  ;
                        34  ; Variable suchen, nicht neu anlegen
                        35  ;
0310: A9 40             36  START   LDA #$40        ;nicht neu
0312: 85 14             37          STA SUBFLAG     ;einrichten
0314: 20 E3 DF          38          JSR PTRGET      ;1. Variable
                        39  ;
                        40  ; Nur eindimensionale Felder
                        41  ;
0317: A0 04             42          LDY #$04
0319: B1 9B             43          LDA (LOWTR),Y   ;Dimensionen
031B: 49 01             44          EOR #$01        ;nur 1 erlaubt
031D: F0 06             45          BEQ OKAY
031F: 20 57 03          46          JSR ENDE        ;Flagge zurück
0322: 4C 96 E1          47          JMP BSSERR      ;Bad Subscript Error
                        48  ;
                        49  ; Prüfen, ob Zeichenkette
                        50  ;
0325: A5 81             51  OKAY    LDA VARNAM      ;1. Zeichen
0327: 45 82             52          EOR VARNAM+1    ;2. Zeichen
0329: 30 06             53          BMI OKAY1       ;String
032B: 20 57 03          54          JSR ENDE        ;Flagge zurück
032E: 4C 76 DD          55          JMP MISMTCH     ;Fehlermeldung
                        56  ;
                        57  ; Vergleichsflagge setzen für < =
                        58  ;
0331: A9 06             59  OKAY1   LDA #$06        ;<=
0333: 85 16             60          STA CMPFLG
                        61  ;
                        62  ; Auf Variable mit Index 0 zeigen
                        63  ;
0335: A5 9B             64          LDA LOWTR
0337: 18                65          CLC
0338: 69 07             66          ADC #$07        ;auf 1. Wert
033A: 85 04             67          STA LINKS       ;zeigen
033C: A5 9C             68          LDA LOWTR+1
033E: 69 00             69          ADC #$00        ;Übertrag
0340: 85 05             70          STA LINKS+1
                        71  ;
                        72  ; Feldgröße bestimmen. Wert-1
                        73  ; ist die Anzahl der Durchläufe
                        74  ;
0342: C8                75          INY
0343: B1 9B             76          LDA (LOWTR),Y   ;Anzahl Hi
0345: 85 01             77          STA ZAHL+1
```

```
0347: C8          78              INY
0348: B1 9B       79              LDA (LOWTR),Y  ;dito Lo
034A: 38          80              SEC
034B: E9 01       81              SBC #$01       ;-1
034D: 85 00       82              STA ZAHL
034F: B0 02       83              BCS NODEC
0351: C6 01       84              DEC ZAHL+1
0353: 05 01       85   NODEC      ORA ZAHL+1
0355: D0 07       86              BNE LOOP1
                  87   ;
                  88   ; Flagge zurücksetzen, dann nächster Befehl
                  89   ;
0357: A9 00       90   ENDE       LDA #$00
0359: 85 14       91              STA SUBFLAG
035B: 4C 95 D9    92              JMP DATA       ;ENDE !!
                  93   ;
                  94   ; Äußere Schleife
                  95   ;
035E: A6 01       96   LOOP1      LDX ZAHL+1     ;Hi
0360: A4 00       97              LDY ZAHL       ;Lo
0362: 84 02       98              STY COUNT
0364: 86 03       99              STX COUNT+1
                  100  ;
0366: A5 04       101             LDA LINKS
0368: A6 05       102             LDX LINKS+1
                  103  ;
                  104  ; Innere Schleife
                  105  ;
036A: 18          106  LOOP2      CLC
036B: 69 03       107             ADC #$03
036D: 85 A0       108             STA MFAC+3     ;merken
036F: 85 06       109             STA RECHTS
0371: 8A          110             TXA
0372: 69 00       111             ADC #$00       ;ev. Übertrag
0374: 85 A1       112             STA MFAC+4
0376: 85 07       113             STA RECHTS+1
0378: A5 04       114             LDA LINKS
037A: A6 05       115             LDX LINKS+1
037C: 85 A8       116             STA SFAC+3
037E: 86 A9       117             STX SFAC+4
                  118  ;
0380: 20 7D DF    119             JSR STRCMP     ;vergleichen
0383: A5 9D       120             LDA MFAC
0385: D0 0F       121             BNE NOSWAP
                  122  ;
                  123  ; Variablen-Deskriptor tauschen
                  124  ;
0387: A0 02       125             LDY #$02
0389: B1 04       126  SWAP       LDA (LINKS),Y
038B: AA          127             TAX            ;retten
038C: B1 06       128             LDA (RECHTS),Y
038E: 91 04       129             STA (LINKS),Y
0390: 8A          130             TXA
0391: 91 06       131             STA (RECHTS),Y
```

```
0393: 88        132             DEY
0394: 10 F3     133             BPL SWAP
                134 ;
                135 ; Schleifenenden, Zähler heruntersetzen
                136 ;
0396: A5 02     137 NOSWAP      LDA COUNT
0398: D0 02     138             BNE INNEN       ;innere Schleife
039A: C6 03     139             DEC COUNT+1
039C: C6 02     140 INNEN       DEC COUNT
039E: A5 02     141             LDA COUNT
03A0: 05 03     142             ORA COUNT+1
03A2: F0 06     143             BEQ AUSSEN
03A4: A5 06     144             LDA RECHTS
03A6: A6 07     145             LDX RECHTS+1    ;hi
03A8: D0 C0     146             BNE LOOP2       ;immer
                147 ;
03AA: A5 04     148 AUSSEN      LDA LINKS
03AC: 18        149             CLC
03AD: 69 03     150             ADC #$03
03AF: 85 04     151             STA LINKS
03B1: 90 02     152             BCC ENDTEST
03B3: E6 05     153             INC LINKS+1     ;Übertrag
03B5: A5 00     154 ENDTEST     LDA ZAHL
03B7: D0 02     155             BNE ET
03B9: C6 01     156             DEC ZAHL+1
03BB: C6 00     157 ET          DEC ZAHL
03BD: A5 00     158             LDA ZAHL
03BF: 05 01     159             ORA ZAHL+1
03C1: F0 94     160             BEQ ENDE
03C3: D0 99     161             BNE LOOP1
```

Das BASIC-Programm zeigt die Leistungsfähigkeit und die Syntax von BSORT.

BSORT.DEMO

```
1   REM  *** SORTIER-DEMONSTRATION ***
5   CLEAR : TEXT
10  PRINT  CHR$ (4);"BRUN BSORT.OBJ"
20  HOME : VTAB 5: INPUT "ANZAHL DER ZU SORTIERENDEN
    STRINGS? ";N
30  DIM A$(N)
40  B$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
50  HOME : VTAB 5: FLASH : PRINT "STRINGS WERDEN
    ERZEUGT": NORMAL
60  FOR I = 1 TO N: FOR J = 1 TO 10
70  A$(I) = A$(I) +  MID$ (B$, INT ( RND (1) * 26) + 1,1)
80  NEXT J,I
90  HOME : FOR I = 1 TO N: PRINT A$(I): NEXT : PRINT
100  INPUT "‹RETURN› SORTIERBEGINN";C$
110  & A$(1)
170  PRINT : INPUT "‹RETURN› FUER SORTIERTE LISTE";C$
180  HOME : VTAB 5: FOR I = 1 TO N
190  PRINT I;: POKE 36,5: PRINT A$(I): NEXT I
```

Zum Aufruf und zur Parameterübergabe benutzen wir wieder den Ampersand-Vektor, der in den Zeilen 26 bis 32 initialisiert wird. Jeder Aufruf führt dann zu START in Zeile 36. Vor der Suche der Variablen mittels PTRGET wird SUBFLAG ($0014) auf $40 (Bit 6) gesetzt. Das verhindert, daß PTRGET die bezeichnete Variable neu anlegt, falls sie noch nicht existiert, denn wir wollen nur vorhandene Felder sortieren. Da BSORT nur für eindimensionale Felder geeignet ist, wird in Zeile 40/41 die Dimensionsanzahl geprüft und im Fehlerfall die Meldung „BAD SUBSCRIPT ERROR" ausgegeben, nachdem zuvor noch die SUBFLAG zurückgesetzt wurde.
Da BSORT nur Zeichenketten bearbeiten kann, muß geprüft werden, ob das aufgerufene Variablenfeld einen String bezeichnet. Dazu werden die von PTRGET in VARNAM abgelegten ersten beiden Variablenzeichen miteinander EOR-verknüpft. Bei einer Zeichenkette ist Bit 7 im ersten Byte gelöscht und im zweiten Byte gesetzt. Durch Exclusiv-ODER verknüpft erhalten wir ein gesetztes Bit 7, also ein negatives Ergebnis. Bei den reellen Zahlen und den Ganzzahlen sind dagegen die Bits 7 entweder beide gesetzt oder beide gelöscht, sodaß die Verknüpfung immer ein gelöschtes Bit 7 und damit ein positives Ergebnis liefert. Unser Test verläuft also ganz einfach und endet im Fehlerfall in der Meldung „TYPE MISMATCH ERROR".
In den Zeilen 59/60 wird die Vergleichflagge CMPFLG ($0016) auf $06 gebracht, um den Vergleichstyp <= festzulegen. Wir gehen darauf noch ein.
Da wir von „Links" nach „Rechts" durch unser Feld gehen wollen, merken wir uns die Adresse des ersten Elements in LINKS und LINKS+1. Beim Bubble-Sort benötigen wir bei n Elementen n-1 Durchläufe. Wir lesen deshalb die Feldgröße aus dem Variablenkopf, subtrahieren 1 und merken uns das Ergebnis in ZAHL und ZAHL+1. In den Zeilen 85 und 86 können Sie sehen, wie man schnell eine 2-Byte-Zahl daraufhin prüfen kann, ob sie Null ist. Der Akkumulator enthält noch das Hi-Byte. Er wird mit dem Lo-Byte ODER-verknüpft (ORA ZAHL+1). Nur wenn beide Bytes $00 sind, wird die Zero-Flag gesetzt, und das Programm fällt durch die Verzweigung nach ENDE durch. Ist ein Byte von Null verschieden, verzweigen wir in die äußere Schleife LOOP1.
Zum Vergleich der Zeichenketten wollen wir die Applesoft-Routine STRCMP ($DF7D) benutzen. In MFAC+3 und MFAC+4 (im folgenden als (MFAC) bezeichnet) muß dazu die Adresse des einen Stringdeskriptors eingetragen werden. SFAC+3 und SFAC+4 (als (SFAC) bezeichnet) müssen auf den anderen Stringdeskriptor zeigen. Die Art des Vergleichs wird durch CMPFLG festgelegt. Dabei gilt:
(SFAC) < (MFAC) ⇒ 4
(SFAC) = (MFAC) ⇒ 2
(SFAC) > (MFAC) ⇒ 1.

Sie können auch zwei Grundoperationen verbinden: < und = ergibt 4 + 2 = 6. Genau diesen Wert haben wir deshalb vorhin nach CMPFLG geschrieben.
Ist das Ergebnis des Vergleichs „wahr", so trägt STRCMP eine 1 in MFAC ein, für „falsch" wird eine 0 nach MFAC geschrieben (genau gesagt: 81 80 00 00 00 00 für 1 und 00 00 00 00 00 00 für 0).
Der Anfang der inneren Schleife LOOP2 tut nichts weiter, als das linke (feste) Element nach (SFAC) und das rechte Element nach (MFAC) einzutragen. Bei jedem Durchgang durch die innere Schleife wird (MFAC) auf das nächste Element weitergesetzt, bis wir das Feldende erreicht haben. Ist das linke Element (SFAC) <= dem rechten Element (MFAC), dann stimmt die Reihenfolge schon, und wir brauchen beide Elemente nicht zu vertauschen. Bei dem Ergebnis „wahr" muß also der Tausch übersprungen werden. „Wahr" bedeutet eine 1 im MFAC. Wird der Exponent von MFAC in den Akkumulator geladen (Zeile 120), so führt „wahr" zu einem Löschen des Zero-Bits im Statusregister. „Wahr" ist also identisch mit einem „NOT EQUAL TO ZERO". Daraus folgt, daß unser Verzweigungsbefehl zum Umgehen des Austausches „BNE" heißen muß.
Die Zeilen 125 bis 133 sollten Ihnen bekannt vorkommen. Es handelt sich um die Routine SWAP, die allerdings hier nur die **drei** Bytes des Deskriptors umsetzen muß, die bei einem Feld in der Tabelle stehen.
Wenn das rechte Element am Feldende angelangt ist, fallen wir in die äußere Schleife. Diese setzt das linke Element um eine Position weiter und führt zurück in die innere Schleife, bis auch das linke Element das Feldende erreicht und wir mit dem Sortieren fertig sind. Das Feldende wird allerdings dabei nicht direkt getestet. Vielmehr laufen für die innere und äußere Schleife je ein Zähler (ZAHL und COUNT) mit. Ist COUNT gleich Null, ist die innere Schleife abgelaufen, ist dagegen auch ZAHL gleich Null, so ist das ganze Programm beendet. Da sowohl ZAHL als auch COUNT 2-Byte-Zähler sind, ist das Herunterzählen und Testen auf Null ein klein wenig komplizierter als bei einfachen Zählbytes.

STRCMP $DF7D
Vergleicht zwei Zeichenketten miteinander. Zeiger auf 1. String in SFAC+3/4, Zeiger auf 2. String in MFAC+3/4. CMPFLG ($0016) gibt die Art des Vergleichs an. Ist der Vergleich „wahr", wird MFAC auf 1 gesetzt, andernfalls auf 0.

Eingabe:
SFAC+3/4 = Zeiger auf 1. String-Deskriptor
MFAC+3/4 = Zeiger auf 2. String-Deskriptor
CMPFLG
(SFAC) > (MFAC) ⇒ 1
(SFAC) = (MFAC) ⇒ 2
(SFAC) >= (MFAC) ⇒ 3
(SFAC) < (MFAC) ⇒ 4
(SFAC) <> (MFAC) ⇒ 5
(SFAC) <= (MFAC) ⇒ 6

Ausgabe:
MFAC = 1 (wahr), MFAC = 0 (falsch)
VALTYP ($0011) = 0

BSORT eignet sich nur für Zeichenketten. Wollen Sie Zahlen miteinander vergleichen, so bietet uns Applesoft eine weitere Routine an, die genau wie STRCMP arbeitet, aber nur Fließkomma-Zahlen vergleicht, die komplett in MFAC und SFAC stehen müssen. Sie können zu diesem Transport z.B. MOVFM ($EAF9) und MOVSM ($E9E3) benutzen.

COMP $DF6A
Vergleicht zwei Fließkomma-Zahlen miteinander, die im SFAC und MFAC stehen. CMPFLG ($0016) gibt die Art des Vergleichs an. Ist der Vergleich „wahr", wird MFAC auf 1 gesetzt, andernfalls auf 0.

Eingabe:
SFAC = 1. FP-Zahl
MFAC = 2. FP-Zahl
CMPFLG
SFAC > MFAC ⇒ 1
SFAC = MFAC ⇒ 2
SFAC >= MFAC ⇒ 3
SFAC < MFAC ⇒ 4
SFAC <> MFAC ⇒ 5
SFAC <= MFAC ⇒ 6

Ausgabe:
MFAC = 1 (wahr), MFAC = 0 (falsch)

Wenn Sie ein universelleres Sortierprogramm benötigen, finden Sie im „Peeker 1/86" eine Quicksort-Routine von H. Grumser, die noch ca. 10-mal schneller als BSORT ist. Sie hat allerdings den Fehler, SUBFLAG nicht wieder auf $00 zu setzen, weshalb z.B. nachfolgende INPUT-Befehle abstürzen. Mit Ihren jetzigen Kenntnissen wird Ihnen die Korrektur des QUICKSORT aber nicht mehr schwerfallen.

7. Kleinvieh macht auch Mist

Bevor wir uns an die letzten drei großen Projekte dieses Buches machen, wollen wir die Anwendung unserer bisherigen Kenntnisse mit ein paar praktischen Programmen üben.

7.1 Konstant im Wandel

Die Umwandlung zwischen Zahlen mit verschiedener Zahlenbasis gehört zu den alltäglichen Aufgaben eines Assembler-Programmierers. Deshalb ist in ASSESSOR auch eine entsprechende Routine eingebaut (die Fragezeichen-Funktion), die Sie ständig aufrufen können. Das Programm HEX-DEZ können Sie dagegen z.B. benutzen, wenn Sie BASIC-Programme schreiben. Es wird in die unter BASIC freie Seite 3 geladen und mit dem Ampersand-Vektor direkt aufgerufen. Dabei benutzen wir wieder zahlreiche ROM-Routinen, die fast die gesamte Arbeit für uns verrichten.

HEX-DEZ

```
1     ;**********************
2     ;                     *
3     ; HEX - DEZ CONVERT  *
4     ;                     *
5     ;**********************
6                 ORG $300
7     ;
8     A2L         EQU $003E
9     A2H         EQU $003F
10    LINNUM      EQU $0050
11    CHRGET      EQU $00B1
12    TXTPTR      EQU $00B8
13    DOSWRM      EQU $03D0
14    AMPER       EQU $03F5
15    FRMEVL      EQU $DD7B
16    SYNERR      EQU $DEC9
17    GETADR      EQU $E752
```

```
                        18   LINPRT    EQU $ED24
                        19   PRNTAX    EQU $F941
                        20   GETNUM    EQU $FFA7
                        21   ;
0300: A9 4C             22             LDA #$4C
0302: 8D F5 03          23             STA AMPER
0305: A9 12             24             LDA #<START
0307: 8D F6 03          25             STA AMPER+1
030A: A9 03             26             LDA #>START
030C: 8D F7 03          27             STA AMPER+2
030F: 4C D0 03          28             JMP DOSWRM
                        29   ;
0312: C9 44             30   START     CMP #'D'
0314: D0 0E             31             BNE NODEZ
                        32   ; Dezimal nach Hexadezimal
0316: 20 B1 00          33             JSR CHRGET      ;1. Ziffer lesen
0319: 20 7B DD          34             JSR FRMEVL      ;Ausdruck -> FAC
031C: 20 52 E7          35             JSR GETADR      ;FAC -> LINNUM,A-Y
031F: A6 50             36             LDX LINNUM      ;Lo
0321: 4C 41 F9          37             JMP PRNTAX      ;hex. ausgeben
                        38   ;
0324: C9 48             39   NODEZ     CMP #'H'
0326: F0 03             40             BEQ HEXDEZ
0328: 4C C9 DE          41             JMP SYNERR      ;ungültig
                        42   ;
032B: A2 00             43   HEXDEZ    LDX #00         ;init.
032D: A4 B8             44             LDY TXTPTR      ;merken und
032F: C8                45             INY             ;auf 1. Ziffer zeigen
0330: 09 80             46   MASKE     ORA #$80        ;Bit7 setzen
0332: 81 B8             47             STA (TXTPTR,X)  ;X = 0 !
0334: 20 B1 00          48             JSR CHRGET      ;nächstes Zeichen
0337: D0 F7             49             BNE MASKE       ;0 = Ende
0339: 20 A7 FF          50             JSR GETNUM      ;ASCII -> HEX
033C: A5 3F             51             LDA A2H         ;Hi
033E: A6 3E             52             LDX A2L         ;Lo
0340: 4C 24 ED          53             JMP LINPRT      ;dez. ausgeben
```

Der Ampersand-Vektor wird in der uns schon bekannten Art auf START gesetzt. Wenn wir das Programm aufrufen wollen, geben wir ein & ein, gefolgt von „D" für eine Dezimalzahl oder „H" für eine HEX-Zahl. Danach steht die Zahl selber. Ihr Wertebereich umfaßt 0 bis 65535 ($0000 – $FFFF). Beim Auftreten des &-Zeichens springt der BASIC-Interpreter über $03F5 nach START, wobei sich das Zeichen nach dem & (hier also D oder H) im Akkumulator befindet. Wenn es ein „D" ist, gelangen die Zeilen 32 bis 37 zur Ausführung. JSR CHRGET stellt den TXTPTR auf das folgende Zeichen und holt es in den Akkumulator. Dies ist die Vorbedingung für die Routine FRMEVL ($DD7B), die einen beliebigen Ausdruck an der TXTPTR-Stelle auswertet und nach MFAC schreibt. FRMEVL gehört zu den mächtigsten Teilen von Applesoft und verarbeitet auch beliebige Kombinationen aus Konstanten und Variablen. 25*A-10 wäre also z.B. erlaubt. Ist der Ausdruck

numerisch, steht in MFAC die ungepackte Fließkomma-Zahl. Ist der Ausdruck eine Zeichenkette, steht ein Zeiger auf den Deskriptor im dritten und vierten Byte des MFAC. GETADR ($E752) wandelt den MFAC in eine Ganzzahl in LINNUM ($0050/51) bzw. im Y-Register und im Akkumulator um. PRNTAX ($F941) gibt schließlich diese Ganzzahl hexadezimal über COUT aus. Dazu muß das Hi-Byte im Akku und das Lo-Byte im X-Register stehen. Das X-Register wird aus LINNUM geladen, der Akku ist bereits korrekt gesetzt.

FRMEVL $DD7B
Wertet beliebigen Ausdruck eines Typs aus.

Eingabe:
TXTPTR zeigt auf 1. Zeichen des Ausdrucks

Ausgabe:
TXTPTR zeigt hinter den Ausdruck
a) Ausdruck ist numerisch
 MFAC = Fließkomma-Zahl des Ausdrucks
 VALTYP ($0011) = $00
b) Ausdruck ist Zeichenkette
 MFAC+3/4 ($00A0/A1) enthält Zeiger auf den Deskriptor
 VALTYP ($0011) = $FF

GETADR $E752
MFAC (<65536) wird in 4-stellige Hexzahl ohne Vorzeichen in LINNUM umgewandelt.

Eingabe: MFAC enthält Fließkomma-Zahl

Ausgabe:
LINNUM ($0050/51) = Ganzzahl-Anteil von MFAC
Akku = Hi-Byte, Y-Reg = Lo-Byte

PRNTAX $F941
Akkumulator und X-Register werden als HEX-Zahl über COUT ausgegeben.

Eingabe:
Akku = Hi-Byte der Zahl
X-Reg = Lo-Byte der Zahl

Ausgabe: Akku verändert

PRNTYX $F940
Y- und X-Register werden als HEX-Zahl über COUT ausgegeben.

Eingabe:
Y-Reg = Hi-Byte der Zahl
X-Reg = Lo-Byte der Zahl

Ausgabe: Akku verändert

War der Kennbuchstabe ein „H", werden die Zeilen 43 bis 53 ausgeführt. In jedem anderen Fall wird „SYNTAX ERROR" ausgegeben.

SYNTERR $DEC9
Gibt „SYNTAX ERROR" aus

Eingabe: –

Ausgabe: Fehlermeldung über COUT. Macht BASIC-Warmstart oder springt in eine ONERR-Behandlung.

Da wir unser Programm aus einer BASIC-Umgebung aufrufen, wird unsere Eingabezeile von Applesoft bearbeitet, d.h. sie wird bei einer Direkteingabe mit gelöschtem Bit 7 im Tastaturpuffer abgelegt. Die Zeilen 43 bis 49 setzen wieder

Bit 7, bis sie auf ein Zeilenende ($00) oder ein Befehlsende (: = $22) stoßen. GETNUM ($FFA7) wandelt die ASCII-kodierte Eingabe in eine 2-Byte HEX-Zahl in A2L/H ($003E/3F) um. LINPRT ($ED24) schließlich gibt den Inhalt von Akkumulator (Hi) und X-Register (Lo-Byte) über COUT als Dezimalzahl aus.

GETNUM $FFA7
Wandelt eine 2 bis 4-stellige HEX-Zahlenkette (ASCII-kodiert mit Bit 7 gesetzt) in eine 1-2 Byte HEX-Zahl in A2L und A2H.

Eingabe:
Zeichenkette im Tastaturpuffer ab $0200, Bit 7 gesetzt
Y-Reg = Position im Puffer (normal auf $00 setzen)

Ausgabe:
A2L/H ($003E/3F) enthält HEX-Zahl (Lo/Hi).

LINPRT $ED24
Gibt 4-stellige HEX-Zahl dezimal über COUT aus.

Eingabe:
X-Reg = Hi-Byte, Akku = Lo-Byte

Ausgabe: –

7.2 Ein Spürhund für Adressen

Eine der sinnvollsten Arten, die Assembler-Programmierung zu erlernen, ist das Disassemblieren und Verstehen von fremden Programmen. Dabei kann es sich um die eingebauten ROMs handeln oder um beliebige ladbare Maschinenprogramme, die irgendetwas für Sie Interessantes tun. Der zunächst schwierigste Teil bei diesem „Schnuppern" in fremden Gedankengängen ist es, einen Überblick über die verschiedenen Teile eines Gesamtprogramms zu erhalten. In

der Regel macht man sich daran, kleine Teile, die leicht verständlich sind, zu disassemblieren und zu kommentieren. Wenn Sie so einen kleinen Teil verstanden haben, ist es sicher interessant zu wissen, welche übrigen Teile des Gesamtprogramms die von Ihnen verstandene Unterroutine benutzen. Sie müssen dazu das gesamte Programm nach Aufrufen für dieses Unterprogramm durchsuchen. Da dies eine sehr zeitaufwändige Tätigkeit sein kann, schreiben wir uns für eine automatische Suche ein kleines Programm. Dieses durchkämmt wie ein Spürhund einen von Ihnen angegebenen Speicherbereich und listet Ihnen alle Zeilen, die einen Aufruf der von Ihnen spezifizierten Adresse enthalten.

CROSSREFERENZ

```
                      1   ;*******************************
                      2   ;  Crossreferenz für Adressen  *
                      3   ;                               *
                      4   ; (C) 1985  Dr. Jürgen Kehrel   *
                      5   ;*******************************
                      6   ;
                      7   ; Start mit BRUN unter DOS 3.3
                      8   ; Holt seine Ladeadresse aus dem
                      9   ; DOS und initialisiert CTRL-Y
                     10   ;
                     11   ; * Adresse<VON.BIS(CTRL-Y)(CR)
                     12   ; z.B.:FDED<800.1000(CTRL-Y)(CR)
                     13   ; Findet keine Zero-Page-Adressen
                     14   ;
                     15   LENGTH    EQU $002F
                     16   PCL       EQU $003A
                     17   PCH       EQU $003B
                     18   A1L       EQU $003C
                     19   A1H       EQU $003D
                     20   A4L       EQU $0042
                     21   A4H       EQU $0043
                     22   USRADR    EQU $03F8
                     23   BLOADADR  EQU $AA72       ;DOS 3.3
                     24   KBD       EQU $C000
                     25   STROBE    EQU $C010
                     26   INSDS2    EQU $F88C
                     27   INSTDSP   EQU $F8D0
                     28   PCADJ3    EQU $F956
                     29   NXTA1     EQU $FCBA
                     30   CROUT     EQU $FD8E
                     31   MONZ      EQU $FF69
                     32   ;
                     33             ORG $300
                     34   ;
0300: AD 72 AA       35   INIT      LDA BLOADADR
0303: 18             36             CLC
0304: 69 14          37             ADC #START-INIT
0306: 8D F9 03       38             STA USRADR+1
0309: AD 73 AA       39             LDA BLOADADR+1
030C: 69 00          40             ADC #$00
030E: 8D FA 03       41             STA USRADR+2
0311: 4C 69 FF       42             JMP MONZ
```

```
                        43  ;
0314: A0 00             44  START   LDY #$00
0316: B1 3C             45          LDA (A1L),Y
0318: C5 42             46          CMP A4L         ;mit Lo-Byte vergl.
031A: D0 1E             47          BNE RELATIV
031C: C8                48          INY             ;nächstes Byte
031D: B1 3C             49          LDA (A1L),Y
031F: C5 43             50          CMP A4H         ;mit Hi-Byte vergl.
0321: D0 17             51          BNE RELATIV     ;nicht gefunden
                        52  ;
                        53  ; Überprüfe, ob 3-Byte Befehl
                        54  ;
0323: A6 3D             55          LDX A1H         ;Adresse - 1
0325: A4 3C             56          LDY A1L         ;zeigt auf Opcode
0327: D0 01             57          BNE NODEC       ;kein Übertrag
0329: CA                58          DEX
032A: 88                59  NODEC   DEY
032B: 86 3B             60          STX PCH
032D: 84 3A             61          STY PCL
032F: A2 00             62          LDX #$00        ;init.
0331: 20 8C F8          63          JSR INSDS2
0334: A5 2F             64          LDA LENGTH
0336: C9 02             65          CMP #$02        ;3-Byte Befehl
0338: F0 26             66          BEQ FUND        ;OK. anzeigen
                        67  ;
033A: A0 00             68  RELATIV LDY #$00        ;Byte erneut
033C: B1 3C             69          LDA (A1L),Y
033E: 29 1F             70          AND #$1F        ;Trick vom alten
0340: C9 10             71          CMP #$10        ;Monitor für Branch
0342: D0 2F             72          BNE WEITER
                        73  ;
                        74  ; Setze Offset zur Adresse um
                        75  ;
0344: A4 3C             76          LDY A1L
0346: A6 3D             77          LDX A1H
0348: 84 3A             78          STY PCL
034A: 86 3B             79          STX PCH
034C: A0 01             80          LDY #$01        ;Offsetbyte lesen
034E: B1 3C             81          LDA (A1L),Y     ;und mit Monitor
0350: 20 56 F9          82          JSR PCADJ3      ;die Adresse-1 in
0353: AA                83          TAX             ;A (Lo) und Y (Hi)
0354: E8                84          INX             ;berechnen, dann
0355: D0 01             85          BNE NOINC       ;korrigieren
0357: C8                86          INY
0358: E4 42             87  NOINC   CPX A4L         ;vergleichen
035A: D0 17             88          BNE WEITER
035C: C4 43             89          CPY A4H
035E: D0 13             90          BNE WEITER      ;nicht gefunden
                        91  ;
                        92  ; Gefundene Stelle disassemblieren
                        93  ;
0360: 20 D0 F8          94  FUND    JSR INSTDSP     ;PCL/PCH OK.
                        95  ;
                        96  ; Pause/Weiter bei Tastendruck
                        97  ;
```

```
0363: 2C 00 C0  98             BIT KBD
0366: 10 0B     99             BPL WEITER      ;kein Tastendruck
0368: 2C 10 C0  100            BIT STROBE      ;löschen
036B: 2C 00 C0  101  PAUSE     BIT KBD         ;neuer Tastendruck?
036E: 10 FB     102            BPL PAUSE       ;Nein
0370: 2C 10 C0  103            BIT STROBE      ;Ja, löschen u. weiter
                104  ;
                105  ; Nächste Adresse, aber $C000-$CFFF auslassen
                106  ;
0373: 20 BA FC  107  WEITER    JSR NXTA1       ;weitersetzen
0376: B0 0C     108            BCS SCHLUSS
0378: A5 3D     109            LDA A1H         ;I/O meiden
037A: C9 C0     110            CMP #$C0
037C: D0 96     111            BNE START
037E: A9 D0     112            LDA #$D0        ;bei $D000 weiter
0380: 85 3D     113            STA A1H
0382: D0 90     114            BNE START       ;immer
                115  ;
0384: 4C 8E FD  116  SCHLUSS   JMP CROUT       ;‹CR› ausgeben, Ende
```

Wir wollen dieses Programm aus dem Monitor heraus aufrufen (also nach CALL-151). Zum Befehlsaufruf und zur Parameterübergabe benutzen wir den Control-Y-Vektor. Er zeigt gewisse Ähnlichkeiten zum Ampersand mit dem Unterschied, daß er nur im Monitor (und nicht von Applesoft aus) benutzt werden kann, und daß er nach $03F8 springt. Außerdem werden alle Zeichen vor <Ctrl-Y> entsprechend einem bestimmten Schema ausgewertet und in Zero-Page (Null-Seiten) Speicherstellen geschrieben.

Unser Programm hat noch eine weitere Eigenschaft: es ist relokativ. Dies heißt, daß es an jeder RAM-Adresse ablauffähig ist. Die meisten Maschinenprogramme können nur ab der Adresse ausgeführt werden, die durch den ORG-Befehl festgelegt wurde. Sie sind nicht verschiebbar (nicht relokativ). Dies liegt daran, daß sie Befehle enthalten, die feste Adressen innerhalb des Codes benutzen. Besonders betroffen davon sind JMPs und JSRs sowie alle Lade- und Speicherbefehle (LD. und ST.), die Adressen innerhalb des Programms betreffen. Wenn sich das Programm nämlich nicht an der durch ORG festgelegten Adresse befindet, stimmen diese Adressen nicht mehr und die Befehle „greifen ins Leere", was in der Regel einen Absturz verursacht. Referenzen auf absolute Adressen **außerhalb** des Programms (z.B. im ROM) sind dagegen kein Hindernis für eine Verschiebung, da diese Adressen immer konstant bleiben, auch wenn das Programm im Speicher verschoben wird. Ein „JSR COUT" ist von überall her ausführbar.
Wenn Sie bei einem Programm schnell einmal sehen müssen, ob es relokativ ist, brauchen Sie sich im Assembler-Listing nur die Befehle anzusehen, die 3 Bytes Objektcode erzeugen. Hiervon ist normalerweise nur das Hi-Byte des Operan-

den interessant. Es steht ganz rechts. Wenn wir durch unser Programm „Crossreferenz" schauen, finden wir die Bytes AA, 03, AA, 03, FF, F8, F9, F8, C0, C0, C0, C0, FC und FD. Mit Ausnahme der beiden „03" bezeichnen sie Hi-Bytes, die außerhalb des Programms liegen. Sie sind einer Verschiebbarkeit also nicht abträglich. Bei den beiden „03" müssen wir uns jetzt noch die Lo-Bytes ansehen: wir finden F9 und FA. Auch sie liegen außerhalb des eigentlichen Programms. Daraus folgt, daß unser Programm keine absolute Adressen in sich selber benutzt, also beliebig im Speicher verschiebbar ist.

Noch eine kleine Anmerkung: jeder JMP zu einem Ziel innerhalb des Programms bindet dieses an seinen Ausführungsort. Wenn das Sprungziel nicht weit entfernt ist, eignet sich deshalb ein „Branch" besser, der relativ adressiert und immer relokativ ist. Besitzer des 65C02 haben es da einfach: „BRA" verzweigt immer zu der angegebenen Adresse. Wenn Sie nur einen 6502 haben, können Sie zunächst eine Statusflagge definiert setzen und diese dann mit einem Branch-Befehl „testen". Besonders beliebt ist die Befehlsfolge

```
CLV
BVC xxxx
```

, da das Löschen des Overflow-Bits normalerweise in einem Programm keinen Schaden anrichtet. Durch diese beiden Befehle erreichen Sie auch eine unbedingte Verzweigung.

Was ist zu tun, wenn das Sprungziel zu weit für einen Branch-Befehl entfernt liegt? Ganz einfach, Sie bauen „Zwischenlandungen" ein, so wie ein Dreispringer auch den Boden zweimal zwischendurch berühren darf. Teilen Sie den gesamten Abstand in mehrere Teilsprünge auf. Da die getestete Flagge sich zwischenzeitlich nicht verändert, reicht es, nur den Branch-Befehl zu wiederholen:

```
     CLV
     BVC EINS    unbedingte Verzweigung
       .
       .
EINS BVC ZWEI    1. Zwischenlandung
       .
       .
ZWEI BVC ZIEL    2. Zwischenlandung
       .
       .
ZIEL XXX YYYY    eigentliches Ziel
```

Unser Programm „Crossreferenz" muß relokativ sein, um jeden erdenklichen Speicherbereich testen zu können. Wäre das Programm an seine ORG-Adresse

gebunden, könnten z.B. nie Crossreferenzen (Querverweise) für den Bereich $0300 bis $0386 erstellt werden, es sei denn, Sie wollten von „Crossreferenz" selber eine Crossreferenz machen.

Die Verschiebbarkeit bringt ein Problem für uns mit: wenn das Progamm über den Control-Y-Vektor gestartet werden soll, muß in den Vektor die tatsächliche Startadresse eingetragen werden. Diese liegt vor dem Lauf des Programms aber nicht fest. Deshalb muß das Programm selber feststellen, wo es sich im Speicher befindet. Dazu wurden verschiedene Techniken entwickelt. Wir gehen hier davon aus, daß das Programm von DOS 3.3 an seine Laufadresse geladen wurde. Auf einem Rechner mit mindestens 48K Hauptspeicher finden wir dann in den Speicherstellen $AA72 und $AA73 die Adresse des letzten „BLOAD" oder „BRUN". Wir nutzen diese Werte, um sie in den Vektor einzutragen. Da diese bei DOS gefundene Ladeadresse die Adresse des ersten Bytes (also von INIT) ist, wir aber das eigentliche Programm bei START starten wollen, zählen wir noch die Länge des INIT-Teils dazu (Zeile 37). Dieses Verfahren klappt nur, wenn Sie das Programm auch an der Ladeadresse laufen lassen. Wenn Sie es dagegen mit „BLOAD" laden, in den Monitor gehen, es verschieben und dann starten, stimmt die Adresse im DOS nicht.
Der Initialisierungsteil schließt mit einem Sprung nach MONZ ($FF69). Dies ist die Einsprungadresse in den Monitor, der Ihnen danach sein Sternchen-Prompt * präsentiert.

Die Syntax von „Crossreferenz" ähnelt der Blockverschiebung des Monitors. Zuerst nennen Sie die gesuchte Adresse in hexadezimaler Form, aber ohne Dollarzeichen. Diese darf nicht auf der Zero-Page (Null-Seite) liegen. Es folgt ein „Kleiner-als"-Zeichen. Daran schließt sich die Adresse des Suchbeginns an, der nach einem Punkt die Adresse des Suchendes folgt. Im Anschluß daran drücken Sie Control-Y (was auf dem Bildschirm nicht zu sehen ist) und die RETURN-Taste:
Adresse<VON.BIS(Ctrl-Y)(CR).

„Crossreferenz" disassembliert Ihnen dann alle Zeilen, die die gesuchte Adresse aufrufen. Das funktioniert auch bei Branch-Befehlen, da hier die relative Adresse in die absolut aufgerufene umgerechnet wird.

Bevor wir uns das Listing näher ansehen, müssen wir noch wissen, was der Monitor mit unseren Eingaben vor dem <Ctrl-Y> macht. Die erste HEX-Zahl (Adresse) wird nach A4L und A4H ($0042/43) gespeichert. Die Adresse nach dem <-Zeichen (VON) landet in A1L und A1H ($003C/3D), während die

Adresse nach dem Punkt (BIS) sich in A2L und A2H ($003E/3F) wiederfindet. In den Zeilen 44 bis 50 wird verglichen, ob sich bei der Adresse, auf die A1L/H zeigt, die Bytefolge der gesuchten Adresse befindet. Ist das nicht der Fall, wird in Zeile 51 verzweigt, um auch relative Adressierungen zu erkennen. Wurde die gesuchte Bytefolge gefunden, muß überprüft werden, ob es sich dabei auch wirklich um einen Operanden handelt. Es könnte ja auch sonst durch Zufall einmal diese Bytekombination entstehen. Es kommen nur Befehle in Frage, die 3 Bytes lang sind. Deshalb wird das Opcode untersucht, das sich vor der gefundenen Bytekombination befindet. Die augenblicklich verglichene Adresse wird in die Indexregister geladen und, um 1 vermindert, nach PCL und PCH ($003B/3C) gespeichert. Sodann wird die Monitorroutine INSDS2 ($F88C) aufgerufen, die unter anderem die Befehlslänge bestimmt. Diese wird immer um 1 kleiner angegeben. Eine 2 in der Speicherstelle LENGTH ($002F) bedeutet also einen 3-Byte-Befehl. Wenn dies zutrifft, haben wir eine Referenz gefunden und geben sie in Zeile 94 mit der Monitorroutine INSTDSP ($F8D0) disassembliert aus.

War der Befehl nicht 3 Bytes lang, testen wir auf einen Branch-Befehl. Die Zeilen 68 bis 71 fischen alle Branch-Opcodes (bis auf BRA!!) heraus. War der Befehl auch kein Branch, gehen wir zur nächsten Speicherstelle weiter (WEITER). Bei einem Branch wird durch die Monitorroutine PCADJ3 ($F956) die tatsächliche Adresse berechnet und mit der gesuchten verglichen.
Nach jeder gefundenen Referenz können wir mit einem beliebigen Tastendruck das Programm anhalten und es mit einem erneuten Tastendruck wieder starten. Die Zeilen 98 bis 103 erledigen dies für uns.
Für das Weitersetzen der Adresse bedienen wir uns einer Monitor-Routine. NXTA1 ($FCBA) erhöht A1L/H um 1 und testet dann, ob die Adresse in A2L/H schon erreicht oder überschritten wurde. Liegt A1L/H noch unterhalb von A2L/H, so ist das Carry-Bit gelöscht, andernfalls gesetzt. Da A2L/H unsere Endadresse enthält, zeigt uns ein gesetztes Carry-Bit das Ende der Suche an.

Noch eine kleine Besonderheit des Apple ist zu berücksichtigen. Der Speicherbereich von $C000 bis $CFFF enthält Softswitches oder Daten von Steckkarten. Es wäre gefährlich bzw. irreführend, diesen Bereich zu durchsuchen. Nach der Adresse $BFFF wird deshalb A1L/H nicht auf $C000, sondern gleich auf $D000 weitergesetzt.

INSDS2 $F88C
Berechnet Länge, Format und Mnemonic-Index für das Opcode, auf das (PCL) zeigt.

Eingabe:
X-Reg = $00
PCL/H ($003A/3B) zeigt auf Opcode

Ausgabe:
FORMAT ($002E) = Adressierungsart in codierter Form
LENGTH ($002F) = Befehlslänge − 1
Akku = Index in der Mnemonic-Tabelle des Monitors

PCADJ3 $F956
Berechnet aus dem Distanzbyte die tatsächlich angesprochene Adresse bei relativer Adressierung.

Eingabe:
Akku = Distanzbyte
PCL/H zeigt auf Distanzbyte

Ausgabe:
Akku = Lo-Byte der Adresse
Y-Reg = Hi-Byte der Adresse

INSTDSP $F8D0
Disassembliert einen Maschinenbefehl durch Ausgabe über COUT.

Eingabe:
PCL/H zeigt auf das Opcode (Befehlsbyte)

Ausgabe:
LENGTH ($002F) = Befehlslänge − 1
Disassembliert wie L-Befehl des Monitors, aber nur 1 Zeile

NXTA1 $FCBA
Setzt A1L/H um 1 weiter und vergleicht mit A2L/H.

Eingabe:
A1L/H ($003C/3D) = zu erhöhender Wert (Lo/Hi)
A2L/H ($003E/3F) = Vergleichswert

Ausgabe:
A1L/H um 1 erhöht
Carry-Bit gesetzt, wenn A1L/H >= A2L/H

Nach einer nur kleinen Änderung ist das Programm unabhängig vom benutzten DOS und deshalb z.B. auch unter ProDOS einsetzbar. Wir finden die aktuelle Adresse durch einen kleinen Trick. Wenn wir ein JSR zu einem bekannten RTS durchführen – z.B. $FF58 im ROM –, dann könnnen wir auf dem Stack die Rücksprungadresse finden. Diese ist um genau 2 größer als die Adresse des JSR. Fügen Sie deshalb im Vereinbarungsteil folgende Zeilen hinzu:

```
STACK EQU $0100
IORTS EQU $FF58 ; festes RTS
```

Den INIT-Teil schreiben Sie wie folgt um:

```
INIT     JSR IORTS     ;auf RTS im ROM
         TSX           ;Stackpointer
         LDA STACK,X   ;Hi-Byte
         STA USRADR+2
         DEX
         LDA STACK,X   ;Lo-Byte
         CLC
         ADC #START-INIT-2
         STA USRADR+1
         BCC MONITOR   ;Übertrag?
         INC USRADR+2  ;Ja
MONITOR  JMP MONZ      ;zum Monitor
```

8. Speicher-Recycling = Overlay

Nachdem Sie sich im Kapitel 6 so sehr mit Variablen quälen mußten, soll jetzt als Belohnung eine praktische Anwendung folgen, die die Möglichkeiten von Applesoft ganz beträchtlich erweitert, besonders wenn Sie gleichzeitig hochauflösende Grafikseiten benutzen wollen.
Aus historischen Gründen liegen beim Apple die beiden HGR-Seiten mitten im heutigen Arbeitsspeicher. Bei kleinen Applesoft-Programmen fällt das nicht weiter auf. Wenn Ihre Programme aber größer werden, passiert es leicht, daß das BASIC-Programm bis in den Bildspeicher reicht, denn der Platz von $0801 (dem normalen Start aller Applesoft-Programme) bis $2000 (dem Beginn von HGR Seite 1) ist nicht gerade sehr groß.
Es gibt nun verschiedene Möglichkeiten, aus diesem Dilemma heraus zu kommen. Eine liegt darin, mit dem BASIC-Programm erst oberhalb der Grafik ($6001) anzufangen, wo mit DOS 3.3 immerhin Platz bis $9600 ist. Eine etwas exotische Möglichkeit wurde vor Jahren in CALL A.P.P.L.E. vorgestellt: das BASIC-Programm wird gesplittet und um die Grafik herum aufgebaut. Diese Lösung ist aber mühsam und unflexibel. Der einfachste Weg ist es, das Programm in mehrere Teile zu gliedern und diese dann nacheinander auszuführen, z.B. über ein EXEC-Programm oder durch direkten Aufruf am Ende des Vorgänger-Programms („RUN XYZ.TEILn").
Diese Lösung hat den Nachteil, daß alle Variablen verloren gehen und zwischen den Modulen nur schwer Informationen übergeben werden können. Einen Schritt besser ist die Verwendung des CHAIN-Programms von der DOS 3.3 SYSTEM MASTER-Diskette. Hier bleiben die Variablen erhalten. Aber auch CHAIN kann immer nur ein Modul **ganz** durch das nächste ersetzen.

Ich habe nun CHAIN analysiert und beträchtlich erweitert. Daraus entstand der „Applesoft-Overlay-Manager". Was kann dieses Programm nun? Meine Überlegungen gingen dahin, daß bestimmte Programmteile viel benötigt werden, andere jedoch nur selten, z.B. wenn sortiert oder gedruckt werden muß. Es sollte also möglich sein, ein „Grundmodul" zu laden, das immer im Speicher

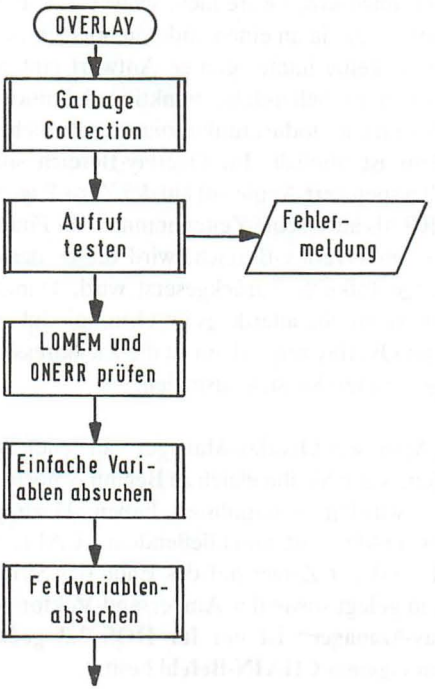
verbleibt. Dieses „Grundmodul" läd seinerseits weitere Programmteile bei Bedarf nach. Es kann sich dabei auch selber teilweise überschreiben. Es können beliebig viele „Zusatzmodule" nachgeladen werden, wobei sie sich überlagern oder ergänzen können. Aus diesem Grunde kann die Zeilennummer angegeben werden, ab der das neue Modul eingeladen wird. Daß sämtliche Variablen erhalten bleiben, ist selbstverständlich.
Zwei kleine Einschränkungen will ich Ihnen nicht verschweigen: Wenn in dem Bereich, der durch das neue Modul überschrieben wird (Overlay-Bereich), ein DEF FN liegt, wird diese Definition nicht gerettet. Sie kann also später nicht benutzt werden. Versuchen Sie es trotzdem, stürzt das Programm ab (auch CHAIN rettet Funktionen nicht). Der Grund ist einfach: Applesoft speichert bei einer Funktionsdefinition deren absolute Lage im Speicher und holt sich bei der Benutzung der Funktion jedesmal die Definition neu aus dem BASIC-Programm. Den Zeiger umzusetzen wäre nicht schwer. Die Frage aber bleibt: wo lasse ich die Definition, die ja an einen anderen Speicherplatz gerettet werden müßte. Da es hierauf keine immer gültige Antwort gibt, wurde ganz darauf verzichtet, Funktionen zu behandeln. Funktionsdefinitionen im nicht überschriebenen Teil des Grundmoduls funktionieren natürlich.
Das zweite Problem ist ähnlich. Im Overlay-Bereich sollte kein ONERR vorkommen. Hierbei speichert Applesoft auf der Zero-Page sowohl die Adresse des gültigen ONERR als auch seine Zeilennummer ab. Findet der Overlay-Manager ein ONERR im Overlay-Bereich, wird dieses deaktiviert, indem die entsprechende Flagge ($00D8) zurückgesetzt wird. Damit sind die meisten Probleme behoben. Wenn Sie allerdings im Grundmodul ein ONERR haben, das auf eine Zeile im Overlay zeigt, dann ist die Zieladresse eventuell gar nicht mehr da. Die Folge können Sie sich ausmalen.

Sie können den „Applesoft-Overlay-Manager" an jeder Stelle ihres BASIC-Programms aufrufen, wenn Sie ihn gleich zu Beginn – noch bevor irgendwelche Variablen definiert wurden – initialisiert haben. Dazu genügt es, ihn mit „BRUN" oder „BLOAD" mit anschließendem „CALL 37632" zu starten, wodurch HIMEM: und der Zeiger auf das Ende des Stringpools (FRETOP) unter das Programm gelegt sowie der Ampersand-Vektor gesetzt werden. **Der „Applesoft-Overlay-Manager" ist nur für DOS 3.3 gedacht und nicht für ProDOS, das einen eigenen CHAIN-Befehl besitzt.**

Der Aufruf erfolgt mit dem Ampersand-Zeichen &, gefolgt von OVL, einem Komma, der Startzeile, ab der das neue Modul eingeladen werden soll, einem weiteren Komma und schließlich dem in Anführungszeichen gesetzten Namen

des aufgerufenen Moduls. Wenn Sie den Programmteil „ABC.TEIL3“ ab Zeile 750 laden wollen, lautet der Befehl:
&OVL,750,“ABC.TEIL3“.
Wichtig ist, daß die Zeilennummern in dem Overlay-Modul mindestens so groß sind wie die im Aufruf angegebene Startzeile. Höhere Zeilennummern schaden nicht, niedrigere können dazu führen, daß eine Nummer zweimal vorkommt. Dann funktionieren Sprünge nicht mehr richtig, da Applesoft nur die zuerst stehende von zwei Zeilen mit gleicher Nummer findet, wenn sie mit GOTO oder GOSUB angesprungen wird.

Bevor wir uns mit der Programmierung auseinandersetzen, folgt jetzt einmal das komplette Programm und anschließend eine kleine Demonstration.



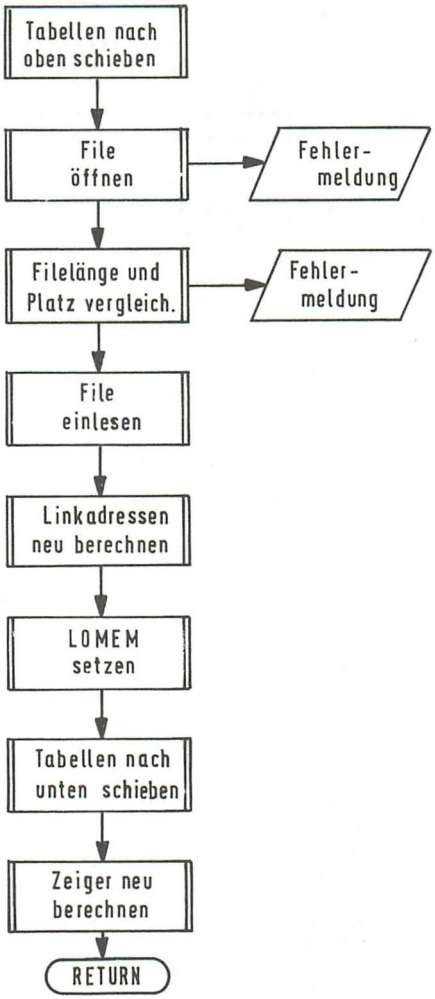


Abb. 10: Vereinfachter Ablaufplan des Applesoft-Overlay-Managers

OVL-MANAGER

```
1   ;*********************************
2   ;  Applesoft-Overlay-Manager   *
3   ; (C) 1985  Dr. Jürgen Kehrel  *
4   ;*********************************
5   ;  Version 1.3    Jan. 86
6   ;
7   ; Benutzt Teile von: CHAIN
8   ; Quelle: Randy Wiggington
9   ; Call A.P.P.L.E. Vol. 2 Nr. 1
10  ;
11  ; Syntax:
12  ; &OVL,‹Zeile›,‹File›
13  ; z.B.  &OVL,500,"MODUL1"
14  ;
15  ; Muß initialisiert werden, bevor
16  ; Stringvariablen angelegt werden!
17  ; Nur für DOS 3.3 !!!
18  ;
19  ; Im Overlay-Bereich darf sich kein
20  ; DEF FN befinden. Ein aktives
21  ; ONERR wird wieder zurückgesetzt.
22  ;
23  ;
24  A1L       EQU $003C
25  A2L       EQU $003E
26  CURBUF    EQU $0040
27  A4L       EQU $0042
28  NXTBUF    EQU $0044
29  INDEX     EQU $005E
30  TXTTAB    EQU $0067
31  VARTAB    EQU $0069
32  ARYTAB    EQU $006B
33  STREND    EQU $006D
34  FRETOP    EQU $006F
35  HIMEM     EQU $0073
36  DATPTR    EQU $007D
37  DSCLEN    EQU $008F
38  HIGHDS    EQU $0094
39  HIGHTR    EQU $0096
40  LOWTR     EQU $009B
41  DSCTMP    EQU $009D
42  FACMO     EQU $00A0
43  PRGEND    EQU $00AF
44  CHRGOT    EQU $00B7
45  ERRFLG    EQU $00D8
46  TEMP      EQU $00EB
47  TEMP2     EQU $00ED
48  LOMEMFLG  EQU $00EF
49  ERRADR    EQU $00F4
50  ;
51  FMGR      EQU $03D6
52  CONNECT   EQU $03EA
53  AMPER     EQU $03F5
54  ;
```

```
                        55   CLRFNBUF  EQU $A095
                        56   ZEROPRMS  EQU $A1AE
                        57   FILEREAD  EQU $A471
                        58   RD2BYTE   EQU $A47A
                        59   NOBUFS    EQU $A6C8
                        60   TOLARGE   EQU $A6CC
                        61   DOSERR    EQU $A6D2
                        62   CMPLPRMS  EQU $A71A
                        63   COPYFN    EQU $A743
                        64   COPYBP    EQU $A74E
                        65   GETBUF    EQU $A764
                        66   FNBUF1    EQU $AA75
                        67   FMPL      EQU $B5BB
                        68   BLTU2     EQU $D39A
                        69   FNDLIN    EQU $D61A
                        70   STKINI    EQU $D683
                        71   NEWSTT    EQU $D7D2
                        72   RESTORE   EQU $D849
                        73   LINGET    EQU $DA0C
                        74   CHKSTR    EQU $DD6C
                        75   FRMEVL    EQU $DD7B
                        76   CHKCOM    EQU $DEBE
                        77   SYNCHR    EQU $DEC0
                        78   SYNERR    EQU $DEC9
                        79   GARBAG    EQU $E484
                        80   FRESTR    EQU $E5FD
                        81   MOVE      EQU $FE2C
                        82   SETKBD    EQU $FE89
                        83   SETVID    EQU $FE93
                        84   ;
                        85   ;
                        86             ORG $9300      ;bei MAXFILES 3
                        87   ;
                        88   ;
                        89   ; Legt HIMEM unter das Maschinen-
                        90   ; programm und setzt den &-Vektor
                        91   ;
9300: D8                92   START     CLD
9301: A9 00             93             LDA #<START
9303: A0 93             94             LDY #>START
9305: 85 73             95             STA HIMEM
9307: 85 6F             96             STA FRETOP
9309: 84 74             97             STY HIMEM+1
930B: 84 70             98             STY FRETOP+1
930D: A9 4C             99             LDA #$4C       ;JMP
930F: 8D F5 03         100             STA AMPER
9312: A9 1D            101             LDA #<OVERLAY
9314: A0 93            102             LDY #>OVERLAY
9316: 8D F6 03         103             STA AMPER+1
9319: 8C F7 03         104             STY AMPER+2
931C: 60               105             RTS
                       106  ;
                       107  ; Hauptprogramm
                       108  ; Packt Strings unter HIMEM
                       109  ;
```

```
931D: 20 84 E4  110 OVERLAY   JSR GARBAG
9320: 20 B7 00  111           JSR CHRGOT
                112 ;
                113 ; Befehlsaufruf "OVL," testen
                114 ;
9323: A2 03     115           LDX #$03
9325: BD A7 95  116 OV1       LDA AUFRUF,X
9328: 20 C0 DE  117           JSR SYNCHR      ;testen
932B: CA        118           DEX
932C: 10 F7     119           BPL OV1
                120 ;
                121 ; Zeilennummer holen und auf
                122 ; das zugehörige Link-Feld zeigen
                123 ;
932E: 20 0C DA  124           JSR LINGET
9331: 20 BE DE  125           JSR CHKCOM      ;folgt Komma?
9334: 20 1A D6  126           JSR FNDLIN
                127 ;
                128 ; Rette Adresse
                129 ;
9337: A5 9C     130           LDA LOWTR+1
9339: A6 9B     131           LDX LOWTR
933B: 8D A6 95  132           STA OVSTART+1
933E: 8E A5 95  133           STX OVSTART
                134 ;
                135 ; Prüfen, ob LOMEM am Ende des
                136 ; Programms liegt. Wenn LOMEM
                137 ; im Programm gesetzt wurde,
                138 ; ist eine Flagge zu setzen.
                139 ;
9341: A5 B0     140           LDA PRGEND+1    ;Hi-Byte
9343: C5 6A     141           CMP VARTAB+1
9345: 90 04     142           BLT SAVLOMEM    ;C = 0 (PRGEND < LOMEM)
9347: A5 AF     143           LDA PRGEND      ;Lo
9349: C5 69     144           CMP VARTAB      ;C = 1 (PRGEND = LOMEM)
934B: 66 EF     145 SAVLOMEM  ROR LOMEMFLG    ;C = 0 (PRGEND < LOMEM)
                146 ;
                147 ; Prüfen, ob ein aktiver ONERR-
                148 ; Befehl im Overlay liegt. Falls
                149 ; ja, wird er desaktiviert.
                150 ;
934D: 24 D8     151           BIT ERRFLG
934F: 10 0C     152           BPL NOONERR
9351: A5 F4     153           LDA ERRADR      ;Lo
9353: C5 9B     154           CMP LOWTR
9355: A5 F5     155           LDA ERRADR+1    ;Hi
9357: E5 9C     156           SBC LOWTR+1
9359: 90 02     157           BCC NOONERR
935B: 46 D8     158           LSR ERRFLG      ;löschen
                159 ;
                160 ; Alle Strings, die zwischen OVSTART
                161 ; und dem Programmende liegen, müssen
                162 ; in den freien Speicher kopiert und
                163 ; ihre Deskriptoren entsprechend ge-
                164 ; ändert werden.
```

```
                    165  ;
                    166  ; Zuerst die einfachen Variablen (Skalaren)
                    167  ;
935D: A9 07         168  NOONERR   LDA #$07       ;7 Bytes Länge
935F: 85 8F         169            STA DSCLEN
9361: A0 00         170            LDY #$00       ;init.
9363: A6 69         171            LDX VARTAB     ;Beginn Skalare
9365: A5 6A         172            LDA VARTAB+1
9367: 86 9D         173            STX DSCTMP     ;zwischenspeichern
9369: 85 9E         174            STA DSCTMP+1
936B: C5 6C         175  SMPLLOOP  CMP ARYTAB+1   ;mit Beginn der Array-
936D: D0 04         176            BNE SAVESMPL
936F: E4 6B         177            CPX ARYTAB     ;Variablen vergleichen
9371: F0 05         178            BEQ SMPLDONE   ;alle Skalare fertig
9373: 20 CC 93      179  SAVESMPL  JSR SMPLSAVE
9376: F0 F3         180            BEQ SMPLLOOP   ;immer
                    181  ;
                    182  ; Und nun auch die Felder (Arrays)
                    183  ;
9378: 86 EB         184  SMPLDONE  STX TEMP       ;entspricht ARYTAB Lo
937A: 85 EC         185            STA TEMP+1     ;entspricht ARYTAB Hi
937C: A9 03         186            LDA #$03       ;Descriptorenlänge
937E: 85 8F         187            STA DSCLEN     ;eintragen
9380: A6 EB         188  NXTARAY   LDX TEMP       ;Adresse zurück Lo
9382: A5 EC         189            LDA TEMP+1     ;dito Hi
9384: C5 6E         190  NXTVAR    CMP STREND+1   ;sind wir fertig?
9386: D0 07         191            BNE ARYSAVE
9388: E4 6D         192            CPX STREND
938A: D0 03         193            BNE ARYSAVE    ;Nein
938C: 4C 32 94      194            JMP DOOVLY     ;JA, lade neuen File
                    195  ;
938F: 86 9D         196  ARYSAVE   STX DSCTMP     ;als Zeiger merken Lo
9391: 85 9E         197            STA DSCTMP+1   ;dito Hi
9393: A0 00         198            LDY #$00
9395: B1 9D         199            LDA (DSCTMP),Y ;Name 1. Byte
9397: C8            200            INY
9398: 51 9D         201            EOR (DSCTMP),Y ;Name 2. Byte
939A: 08            202            PHP            ;Status merken
939B: C8            203            INY
939C: B1 9D         204            LDA (DSCTMP),Y ;Offset Lo
939E: 65 EB         205            ADC TEMP       ;+ aktuelle Adresse
93A0: 85 EB         206            STA TEMP       ;= Beginn der näch-
93A2: C8            207            INY            ;  sten Variablen
93A3: B1 9D         208            LDA (DSCTMP),Y ;Offset Hi
93A5: 65 EC         209            ADC TEMP+1     ;+ aktuelle Adresse
93A7: 85 EC         210            STA TEMP+1     ;= Beginn nächst. Var.
93A9: 28            211            PLP            ;Status zurück
93AA: 10 D4         212            BPL NXTARAY    ;kein String!!
93AC: C8            213            INY
93AD: B1 9D         214            LDA (DSCTMP),Y ;Zahl der Dimensionen
93AF: A0 00         215            LDY #$00
93B1: 0A            216            ASL            ;* 2
93B2: 69 05         217            ADC #$05       ;+ 5 für Kopfteil
93B4: 65 9D         218            ADC DSCTMP     ;+ Startadresse
```

```
93B6: 85 9D     219           STA DSCTMP     ;= 1. Array-String Lo
93B8: AA        220           TAX
93B9: 90 02     221           BCC NJETINC    ;kein Übertrag
93BB: E6 9E     222           INC DSCTMP+1   ;weitersetzen
93BD: A5 9E     223 NJETINC   LDA DSCTMP+1   ;Descriptor-Adr. Hi
93BF: C5 EC     224 ARYLOOP   CMP TEMP+1     ;nächste Variable?
93C1: D0 04     225           BNE RETARRAY   ;retten
93C3: E4 EB     226           CPX TEMP       ;Lo-Byte ebenfalls
93C5: F0 BD     227           BEQ NXTVAR     ;also diese Var. fertig
93C7: 20 D6 93  228 RETARRAY  JSR STRGFND    ;String behandeln
93CA: F0 F3     229           BEQ ARYLOOP    ;immer
                230 ;
                231 ; Alle Strings im Overlay-Bereich
                232 ; mit Länge >0 sind gesucht.
                233 ;
93CC: B1 9D     234 SMPLSAVE  LDA (DSCTMP),Y ;Name 1. Byte
93CE: 30 50     235           BMI WEITER     ;Int-Variable!/Funktion!!
93D0: C8        236           INY            ;Name 2. Byte
93D1: B1 9D     237           LDA (DSCTMP),Y
93D3: 10 4B     238           BPL WEITER     ;Real-Variable!
93D5: C8        239           INY
                240 ;
                241 ; Seiteneinstieg für Arrayvariablen
                242 ;
93D6: B1 9D     243 STRGFND   LDA (DSCTMP),Y ;Länge des Strings
93D8: F0 46     244           BEQ WEITER     ;= Null, übergehen
                245 ;
                246 ; Liegt der String im Overlay ?
                247 ;
93DA: C8        248           INY
93DB: B1 9D     249           LDA (DSCTMP),Y ;Low-Pointer
93DD: AA        250           TAX            ;merken
93DE: C8        251           INY
93DF: B1 9D     252           LDA (DSCTMP),Y ;High-Pointer
93E1: 85 9C     253           STA LOWTR+1    ;zwischenspeichern
93E3: 86 9B     254           STX LOWTR
93E5: EC A5 95  255           CPX OVSTART    ;unter dem Overlay?
93E8: ED A6 95  256           SBC OVSTART+1
93EB: 90 33     257           BCC WEITER     ;kann bleiben
93ED: E4 AF     258           CPX PRGEND     ;im Programm?
93EF: A5 9C     259           LDA LOWTR+1    ;Hi zurück
93F1: E5 B0     260           SBC PRGEND+1
93F3: B0 2B     261           BCS WEITER
                262 ;
                263 ; String liegt im Overlay-Bereich
                264 ; Setze den Deskriptor um und ko-
                265 ; piere den String unter FRETOP.
                266 ;
93F5: 88        267           DEY            ;2 zurück auf Länge
93F6: 88        268           DEY
93F7: B1 9D     269           LDA (DSCTMP),Y ;Länge laden
93F9: 48        270           PHA            ;zwischenspeichern
93FA: 38        271           SEC            ;Subtr. vorbereiten
93FB: A5 6F     272           LDA FRETOP     ;Untergrenze Stringpool
```

```
93FD: 85 94      273             STA HIGHDS      ;Zielende Lo
93FF: F1 9D      274             SBC (DSCTMP),Y ;- Länge
9401: C8         275             INY
9402: 91 9D      276             STA (DSCTMP),Y ;neue Lo-Adresse
9404: 85 6F      277             STA FRETOP      ;Grenze herabsetzen
9406: C8         278             INY
9407: A5 70      279             LDA FRETOP+1    ;Untergrenze Stringpool
9409: 85 95      280             STA HIGHDS+1    ;Zielende Hi
940B: E9 00      281             SBC #$00        ;um Übertrag korrigieren
940D: 91 9D      282             STA (DSCTMP),Y ;neue Hi-Adresse eintragen
940F: 85 70      283             STA FRETOP+1    ;Stringpool-Grenze neu
9411: 68         284             PLA             ;Länge zurück
9412: 18         285             CLC
9413: 65 9B      286             ADC LOWTR       ;Anfangsadresse addieren
9415: 85 96      287             STA HIGHTR      ;Quellende Lo eintragen
9417: A5 9C      288             LDA LOWTR+1
9419: 69 00      289             ADC #$00
941B: 85 97      290             STA HIGHTR+1    ;Quellende Hi eintragen
941D: 20 9A D3   291             JSR BLTU2       ;String verschieben
                 292 ;
                 293 ; Zeige auf den nächsten Eintrag
                 294 ;
9420: A5 8F      295 WEITER      LDA DSCLEN      ;Länge des Deskriptors
9422: 18         296             CLC
9423: 65 9D      297             ADC DSCTMP      ;zum letzen dazuzählen
9425: 85 9D      298             STA DSCTMP      ;neue Adresse eintragen
9427: 90 02      299             BCC NOINC       ;kein Übertrag
9429: E6 9E      300             INC DSCTMP+1    ;Hi-Adresse weiterzählen
942B: A6 9D      301 NOINC       LDX DSCTMP      ;für Bereichsüberprüfung
942D: A5 9E      302             LDA DSCTMP+1    ;jetzt laden
942F: A0 00      303             LDY #$00        ;Z-Flagge setzen
9431: 60         304             RTS             ;zurück, nächster String
                 305 ;
                 306 ; Gesamte Variablentabelle nach
                 307 ; oben unter FRETOP schieben.
                 308 ; Pointer in den Tastaturpuffer retten
                 309 ;
9432: A9 00      310 DOOVLY      LDA #$00
9434: 85 94      311             STA HIGHDS      ;Zielende Lo
9436: 85 9C      312             STA LOWTR+1     ;Quellanfang Hi
9438: 85 97      313             STA HIGHTR+1    ;Quellende Hi
943A: A9 69      314             LDA #$69
943C: 85 9B      315             STA LOWTR       ;Quellanfang Lo
943E: A9 71      316             LDA #$71
9440: 85 96      317             STA HIGHTR      ;Quellende Lo
9442: A9 03      318             LDA #$03
9444: 85 95      319             STA HIGHDS+1    ;Zielende Hi
9446: 20 9A D3   320             JSR BLTU2       ;$69-$70 -› $2F8-$2FF
9449: A2 01      321             LDX #$01
944B: B5 69      322 VARMOVE     LDA VARTAB,X    ;Beginn Skalare
944D: 95 9B      323             STA LOWTR,X     ;= Quellanfang
944F: B5 6D      324             LDA STREND,X    ;Ende der String-Var.
9451: 95 96      325             STA HIGHTR,X    ;= Quellende
9453: B5 6F      326             LDA FRETOP,X    ;Untergrenze Stringpool
```

```
9455: 95 94     327             STA HIGHDS,X    ;= Zielende
9457: CA        328             DEX
9458: F0 F1     329             BEQ VARMOVE
945A: 20 9A D3  330             JSR BLTU2       ;Variablen/Stringzeiger
945D: A5 94     331             LDA HIGHDS      ;neuer Anfang Lo
945F: 85 ED     332             STA TEMP2       ;merken
9461: A6 95     333             LDX HIGHDS+1    ;(neuer Anfang Hi)-1
9463: E8        334             INX             ;korrigieren
9464: 86 EE     335             STX TEMP2+1     ;merken
                336 ;
                337 ; Der zu lesende File muß geöffnet
                338 ; werden, um seine Länge zu bestimmen.
                339 ;
9466: 20 95 A0  340             JSR CLRFNBUF    ;Namenspuffer löschen
                341 ;
                342 ; Filenamen überprüfen
                343 ;
9469: 20 7B DD  344             JSR FRMEVL      ;auswerten
946C: 20 6C DD  345             JSR CHKSTR      ;String?
                346 ;
                347 ; FACM0 enthält Zeiger auf Deskriptor
                348 ;
946F: A0 00     349             LDY #$00
9471: B1 A0     350             LDA (FACM0),Y   ;Länge
9473: AA        351             TAX             ;retten
9474: C8        352             INY
9475: B1 A0     353             LDA (FACM0),Y   ;Lo
9477: 85 9D     354             STA DSCTMP
9479: C8        355             INY
947A: B1 A0     356             LDA (FACM0),Y   ;Hi
947C: 85 9E     357             STA DSCTMP+1
                358 ;
                359 ; (DSCTMP) zeigt auf String. Kopiere
                360 ; ihn in den Filenamenspuffer.
                361 ;
947E: CA        362             DEX             ;0<Länge<31
947F: 30 04     363             BMI ERROR
9481: E0 30     364             CPX #$30
9483: 90 03     365             BLT NAMEOK
9485: 4C C9 DE  366 ERROR       JMP SYNERR      ;Syntax-Error
                367 ;
9488: 8A        368 NAMEOK      TXA             ;X->Y
9489: A8        369             TAY
948A: B1 9D     370 COPY        LDA (DSCTMP),Y
948C: 09 80     371             ORA #$80        ;Bit 7 setzen
948E: 99 75 AA  372             STA FNBUF1,Y
9491: 88        373             DEY
9492: 10 F6     374             BPL COPY
                375 ;
                376 ; kopierten String wieder freisetzen
                377 ;
9494: 20 FD E5  378             JSR FRESTR
                379 ;
                380 ; File-Manager-Parameterliste initialisieren
                381 ;
```

```
9497: 20 89 FE  382           JSR SETKBD      ;DOS aus dem Daten-
949A: 20 93 FE  383           JSR SETVID      ;verkehr ausschließen
949D: 20 AE A1  384           JSR ZEROPRMS
                385 ;
                386 ; Freien DOS-Puffer suchen
                387 ;
94A0: 20 64 A7  388           JSR GETBUF
94A3: A5 45     389           LDA NXTBUF+1
94A5: D0 03     390           BNE BUFOK
94A7: 4C C8 A6  391           JMP NOBUFS
                392 ;
94AA: 85 41     393 BUFOK     STA CURBUF+1    ;jetziger Puffer
94AC: A5 44     394           LDA NXTBUF
94AE: 85 40     395           STA CURBUF
                396 ;
                397 ; Filenamen kopieren, Liste vervollständigen
                398 ;
94B0: 20 43 A7  399           JSR COPYFN
94B3: 20 4E A7  400           JSR COPYBP      ;Pufferzeiger
94B6: 20 1A A7  401           JSR CMPLPRMS    ;Parameter
                402 ;
                403 ; Jetzt File öffnen (nicht anlegen)
                404 ;
94B9: A2 01     405           LDX #$01
94BB: 8E BB B5  406           STX FMPL
94BE: 20 D6 03  407           JSR FMGR
                408 ;
                409 ; Auf Fehler und Filetyp prüfen
                410 ;
94C1: 90 06     411           BCC OKAY
94C3: AD C5 B5  412           LDA FMPL+$A     ;Fehlertyp
94C6: 4C D2 A6  413 FEHLER    JMP DOSERR
                414 ;
94C9: AD C2 B5  415 OKAY      LDA FMPL+$7
94CC: 29 7F     416           AND #$7F        ;Lock-Bit ausblenden
94CE: C9 02     417           CMP #$02
94D0: F0 04     418           BEQ FILEOKAY
94D2: A9 0D     419           LDA #$0D
94D4: D0 F0     420           BNE FEHLER      ;immer
                421 ;
                422 ; File ist offen, bestimme/prüfe Länge
                423 ;
94D6: 20 7A A4  424 FILEOKAY  JSR RD2BYTE
94D9: 18        425           CLC
94DA: 6D A5 95  426           ADC OVSTART
94DD: AA        427           TAX             ;merken
94DE: 98        428           TYA             ;Hi-Byte Länge
94DF: 6D A6 95  429           ADC OVSTART+1
94E2: C5 EE     430           CMP TEMP2+1     ;genug Platz?
94E4: 90 03     431           BCC SIZEOK
94E6: 4C CC A6  432           JMP TOLARGE
                433 ;
                434 ; Jetzt File einlesen
                435 ;
```

```
94E9: AE A5 95  436 SIZEOK    LDX OVSTART    ;Ladeadresse Lo
94EC: AC A6 95  437           LDY OVSTART+1  ; Hi
94EF: 20 71 A4  438           JSR FILEREAD
94F2: 20 EA 03  439           JSR CONNECT    ;DOS anhängen
                440 ;
                441 ; Wenn DATA-Zeiger im Overlay,
                442 ; Restore ausführen . Die
                443 ; Linkadressen neu berechnen
                444 ; (dem AS-ROM entlehnt)
                445 ;
94F5: A5 7D     446           LDA DATPTR
94F7: CD A5 95  447           CMP OVSTART
94FA: A5 7E     448           LDA DATPTR+1
94FC: ED A6 95  449           SBC OVSTART+1
94FF: 90 03     450           BCC NOREST
9501: 20 49 D8  451           JSR RESTORE
                452 ;
9504: 20 83 D6  453 NOREST    JSR STKINI
9507: A5 67     454           LDA TXTTAB     ;Programmanfang
9509: A4 68     455           LDY TXTTAB+1
950B: 85 5E     456           STA INDEX      ;Zeiger
950D: 84 5F     457           STY INDEX+1
950F: 18        458           CLC
9510: A0 01     459 LINKLOOP  LDY #$01
9512: B1 5E     460           LDA (INDEX),Y  ;alte Adresse Hi
9514: F0 1D     461           BEQ LINKEND
9516: A0 04     462           LDY #$04       ;Offset
9518: C8        463 ENDLOOP   INY            ;Zeilenende suchen
9519: B1 5E     464           LDA (INDEX),Y
951B: D0 FB     465           BNE ENDLOOP
951D: C8        466           INY            ;Zeilenlänge
951E: 98        467           TYA
951F: 65 5E     468           ADC INDEX
9521: AA        469           TAX            ;merken
9522: A0 00     470           LDY #$00
9524: 91 5E     471           STA (INDEX),Y  ;eintragen
9526: A5 5F     472           LDA INDEX+1
9528: 69 00     473           ADC #$00       ;ev. Übertrag
952A: C8        474           INY
952B: 91 5E     475           STA (INDEX),Y  ;eintragen
952D: 86 5E     476           STX INDEX
952F: 85 5F     477           STA INDEX+1    ;Zeiger weitersetzen
9531: 90 DD     478           BCC LINKLOOP   ;immer
                479 ;
                480 ; Zeiger herstellen
                481 ;
9533: 18        482 LINKEND   CLC
9534: A5 5E     483           LDA INDEX
9536: 69 02     484           ADC #$02
9538: 85 AF     485           STA PRGEND
953A: 85 69     486           STA VARTAB
953C: A5 5F     487           LDA INDEX+1
953E: 69 00     488           ADC #$00       ;Übertrag?
9540: 85 B0     489           STA PRGEND+1
9542: 85 6A     490           STA VARTAB+1
```

```
                        491  ;
                        492  ; Wenn der alte LOMEM-Wert gerettet
                        493  ; werden sollte, ist dieser zu über-
                        494  ; nehmen, wenn das mit dem neuen
                        495  ; Programmende noch möglich ist.
                        496  ;
9544: 24 EF             497           BIT LOMEMFLG
9546: 30 18             498           BMI NOSAVE
9548: AD F8 02          499           LDA $2F8        ;altes LOMEM
954B: C5 AF             500           CMP PRGEND
954D: AD F9 02          501           LDA $2F9
9550: E5 B0             502           SBC PRGEND+1
9552: 90 0C             503           BCC NOSAVE      ;nicht möglich
                        504  ;
9554: A2 05             505           LDX #$05
9556: BD F8 02          506  SAVELOOP  LDA $2F8,X
9559: 95 69             507           STA VARTAB,X
955B: CA                508           DEX
955C: 10 F8             509           BPL SAVELOOP
955E: 30 18             510           BMI MOVESTR
                        511  ;
                        512  ;
                        513  ; Die Zeiger sind neu zu berechnen
                        514  ;
9560: 38                515  NOSAVE    SEC
9561: A5 69             516           LDA VARTAB      ;neuen VARTAB-Wert
9563: ED F8 02          517           SBC $2F8        ;alten Wert abziehen
9566: 85 9D             518           STA DSCTMP      ;merken
9568: A5 6A             519           LDA VARTAB+1    ;dito für Hi-Byte
956A: ED F9 02          520           SBC $2F9
956D: 85 9E             521           STA DSCTMP+1    ;(Differenz Programmenden)
956F: A2 FA             522           LDX #$FA        ;alter ARYTAB-Wert
9571: 20 94 95          523           JSR ADDDIF      ;neuen Wert berechnen
9574: E8                524           INX             ;insgesamt +2
9575: 20 94 95          525           JSR ADDDIF      ;auch für STREND
9578: A2 01             526  MOVESTR   LDX #$01
957A: BD FE 02          527  COPYLOOP  LDA $2FE,X     ;alter FREETOP
957D: 95 6F             528           STA FRETOP,X    ;umkopieren
957F: 95 3E             529           STA A2L,X       ;Quellbereichs-Ende
9581: B5 ED             530           LDA TEMP2,X     ;Quellbereichs-Anfang
9583: 95 3C             531           STA A1L,X       ;für MOVE eintragen
9585: B5 69             532           LDA VARTAB,X    ;Zielbereichs-Anfang
9587: 95 42             533           STA A4L,X
9589: CA                534           DEX
958A: F0 EE             535           BEQ COPYLOOP
958C: A0 00             536           LDY #$00
958E: 20 2C FE          537           JSR MOVE        ;Variablen/Descr. versch.
                        538  ;
                        539  ; Den nächsten Befehl ausführen
                        540  ;
9591: 4C D2 D7          541           JMP NEWSTT
                        542  ;
9594: 18                543  ADDDIF    CLC
9595: BD 00 02          544           LDA $200,X      ;ARYTAB bzw. STREND Lo
```

```
9598: 65 9D     545             ADC DSCTMP      ;+ Differenz
959A: 95 71     546             STA $71,X       ;Offset +$71 -›› $6B, $6D
959C: E8        547             INX             ;Hi-Byte
959D: BD 00 02  548             LDA $200,X      ;alter Wert
95A0: 65 9E     549             ADC DSCTMP+1    ;+ Differenz
95A2: 95 71     550             STA $71,X       ;Offset +$71 -›› $6C, $6E
95A4: 60        551             RTS             ;zurück
                552  ;
                553  OVSTART    DFS 2
                554  ;
95A7: 2C 4C 56  555  AUFRUF     ASC ',LVO'      ;"OVL,"
```

Diese Demonstration, die einer Demo aus Nibble 4/1984 Seite 78 nachempfunden ist, benutzt nur 2 Overlays, die Teile des Grundmoduls überschreiben. Sie können sich durch das Programm jeweils die Zeilen im Speicher listen lassen, um die aktiven Programmzeilen zu sehen.

OVL-DEMO

```
5   TEXT : PRINT  CHR$ (21): HOME
10  INVERSE : HTAB 12: PRINT " OVERLAY - DEMO ": NORMAL
20  HTAB 6: PRINT "(C) 1986 DR. JUERGEN KEHREL": POKE 34,3
30  PRINT  CHR$ (4)"BLOAD OVL-MANAGER.OBJ": CALL 37632
40  VTAB 4: PRINT "DIESE DEMONSTRATION SOLL IHNEN EINEN"
50  PRINT "KLEINEN EINDRUCK VON DEN MOEGLICHKEITEN"
60  PRINT "DES OVERLAY-MANAGERS VERMITTELN."
70  PRINT : PRINT "DIE PROGRAMMZEILEN AB 500 WERDEN NACH"
80  PRINT "UND NACH ERSETZT, OHNE DASS VORANSTEHEN-";
90  PRINT "DE ZEILEN ODER VARIABLEN VERLOREN GEHEN."
100  GOSUB 450
140  VTAB 12: CALL  - 958
150  PRINT "WIR WERDEN DIESE ZEILEN JETZT AUS-"
160  PRINT "FUEHREN": FOR I = 1 TO 2500: NEXT I
170  GOSUB 500: HOME : VTAB 7: PRINT "WIR LADEN EINE SORTIERROUTINE."
180  PRINT
190  & OVL,500,"SORTER"
200  PRINT "AB ZEILE 500 STEHT JETZT EIN SORTIER-"
210  PRINT "PROGRAMM IM SPEICHER."
220  GOSUB 450: VTAB 12: CALL  - 958
230  PRINT "IHRE EINGABEN WERDEN SORTIERT!"
240  GOSUB 500
250  HOME : VTAB 7: PRINT "WIR LADEN EINE AUSGABEROUTINE NACH."
260  & OVL,500,"PRINTER"
270  PRINT : PRINT "AB ZEILE 500 STEHT JETZT EIN AUSGABE-"
280  PRINT "PROGRAMM IM SPEICHER."
290  GOSUB 450: VTAB 12: CALL  - 958
300  GOSUB 500: PRINT : PRINT "MIT LIST KOENNEN SIE SICH DAS ZUSAMMEN-"
310  PRINT "GESETZTE PROGRAMM ANSEHEN.": POKE 34,0: VTAB 20: END
450  VTAB 12: PRINT "AUGENBLICKLICH LAUTEN DIE ZEILEN SO:"
460  VTAB 13: FOR I = 1 TO 40: PRINT "_";: NEXT
470  LIST 500,600
480  GET C$
490  RETURN
500  REM     EINGABEROUTINE
510  HOME : PRINT "BITTE GEBEN SIE 10 ZEICHENKETTEN EIN:"
```

```
520  FOR I = 1 TO 10: PRINT I;: HTAB 5: INPUT A$(I): NEXT
530  RETURN
```

Overlay-Modul „SORTER“

```
500 REM SORTIERT DATEN IN A$()
510 FOR I = 2 TO 10: PRINT ".";
520 FOR J = 1 TO I
530 IF A$(J) < = A$(I) THEN 550
540 T$ = A$(I):A$(I) = A$(J):A$(J) = T$
550 NEXT J,I: RETURN
```

Overlay-Modul „PRINTER“

```
500  REM    AUSGABE DER DATEN IN A$()
510  FOR I = 1 TO 10
520  PRINT I;: HTAB 5: PRINT A$(I)
530  NEXT I: RETURN
```

Der „Applesoft-Overlay-Manager“ gehört schon in die Klasse der komplexeren Programme. Aus diesem Grunde wurde das Gesamtprogramm in viele kleine Abschnitte eingeteilt, die eine Funktion erfüllen und jeweils eine eigene Überschriftszeile besitzen. Wegen der Länge des Programms ist es nicht möglich, jedes einzelne Byte jetzt noch einmal extra zu beschreiben. Sie sollten die zahlreichen Kommentare im Listing beachten. Diese sind im übrigen nicht deshalb so ausführlich, weil das Programm veröffentlicht wird. Auch meine „privaten“ Schöpfungen versehe ich reichlich mit Bemerkungen, weil ich sonst nach einem Jahr selber nicht mehr durch meine Werke „durchblicke“. Auch wenn Kommentare zunächst viel Tipparbeit bedeuten, machen sie sich bei notwendigen Verbesserungen oder Ergänzungen schnell bezahlt.

Nach dem Aufruf aus dem BASIC-Programm beginnt die Ausführung ab OVERLAY (Zeile 110). Hier wird zunächst zwangsweise die Applesoft-Garbage-Collection aufgerufen, die alle Strings im Stringpool – zwischen FRETOP und HIMEM: – fein säuberlich unterhalb von HIMEM: anordnet sowie alle „Stringleichen“ entfernt. Mit der schon beschriebenen Routine SYNCHR ($DEC0) vergleichen wir, ob dem & die Zeichen „OVL“ folgen. Wenn dies nicht der Fall ist, erhalten wir die Fehlermeldung „SYNTAX ERROR“, andernfalls wird mit CHKCOM ($DEBE) das Komma geprüft und übersprungen. TXTPTR zeigt dann auf das erste Zeichen der Zeilennummer, die mit LINGET ($DA0C) von ihrer ASCII-codierten Form in eine 2-Byte-Hex-Zahl in LINNUM ($00500/51) verwandelt wird. FNDLIN ($D61A) liest diesen Wert und sucht die zugehörige Zeile im Programmspeicher (im BASIC-Programm). Existiert die Zeile nicht, wird statt der Zeilenadresse die Adresse der nächsthö-

heren Zeile oder das Programmende in LOWTR übergeben, falls keine Zeile mehr folgt.

LINGET $DA0C
Wertet eine Zahl (0-63999) an der TXTPTR-Stelle aus und wandelt sie in eine HEX-Zahl ohne Vorzeichen

Eingabe:
TXTPTR zeigt auf 1. Zeichen, das sich auch im Akku befinden muß (z.B. durch JSR CHRGOT). Bit 7 im ASCII-String muß 0 sein.

Ausgabe:
LINNUM ($0050/51) = ausgewertete Zahl (Lo/Hi)

FNDLIN $D61A
Sucht die in LINNUM angegebene Applesoftzeile

Eingabe:
LINNUM ($0050/51) Zeilennummer (Lo/Hi)

Ausgabe:
Carry gelöscht, wenn Zeile gefunden
Carry gesetzt, wenn Zeile nicht gefunden
LOWTR ($009B/9C) Adresse des Zeilenanfangs bzw. der nächst höheren Zeile (ev. Programmende)

Wenn Sie mit hochauflösender Grafik arbeiten, ist es üblich, LOMEM hochzusetzen, damit die Variablen erst oberhalb der Grafikseite(n) beginnen. Nach dem Einladen des Overlays wird das neue LOMEM entweder
a) hinter das neue Programmende gelegt, wenn es vorher auch dort war,
b) auf seinen alten Wert gesetzt, wenn es vorher nicht mit dem Programmende identisch war.

Der Fall a) ist die Normalsituation und recht einfach zu bewerkstelligen. Der Fall b) ist interessanter. Wenn Sie LOMEM hochgesetzt haben, wird dies in den Zeilen 140ff festgestellt und vermerkt. Nach dem Laden des Overlay wird

getestet, ob das neue Programmende unterhalb des alten LOMEM liegt. Ist das der Fall, wird LOMEM wieder hergestellt (Zeilen ab 497). Ist das Programm aber so lang geworden, daß der alte LOMEM-Wert überschritten wird, so wird das neue LOMEM an das Programmende gelegt.
Durch diesen etwas komplizierten Verlauf ist gewährleistet, daß LOMEM niemals fälschlich innerhalb des BASIC-Programms liegt, daß es aber auch nicht in die Grafik rutscht, wenn es vorher darübergelegt worden war.

Die Prüfung eines aktiven ONERR wurde bereits besprochen.
Bevor nun das neue Modul eingelesen werden kann, müssen noch einige Zeichenketten gerettet werden. Applesoft verfrachtet nämlich nicht alle in den Stringpool, sondern setzt den Zeiger in das BASIC-Programm, wenn eine Zeichenkette dort direkt definiert wurde. Nehmen wir ein Beispiel:
10 A$ = "HALLO"
Jetzt würde der Zeiger in den BASIC-Text weisen.
20 B$ = A$ + " SIE DA"
B$ ist zusammengesetzt und wird im Stringpool abgelegt.

Wenn nun einige direkt definierte Zeichenketten im Overlay-Bereich liegen, müssen sie vor dem Laden des neuen Moduls aus dem BASIC-Programm in den Stringpool kopiert werden, um erhalten zu bleiben. Alle Zeilen von 168 bis 305 machen nichts anderes, als sowohl die Tabelle der einfachen als auch der Feldvariablen nach Strings zu durchsuchen, bei denen der Zeiger im Deskriptor in den Overlay-Bereich weist. Wird ein solcher gefunden, wird er unter den bislang letzten String kopiert und FRETOP entsprechend erniedrigt. Da dies immer eine Verschiebung von unten nach oben ist, benutzen wir die Routine BLTU2 ($D39A) dazu.

BLTU2 $D39A
Verschiebt den Speicherbereich von LOWTR (Anfang) bis HIGHTR (altes Ende) nach HIGHDS (neues Ende). Nicht geeignet für Verschiebung nach unten und Überschneidung von Quell- und Zielbereich.

Eingabe:
LOWTR ($009B/9C) = Zeiger auf Quellanfang
HIGHTR ($0096/97) = Zeiger auf Quellende
HIGHDS ($0094/95) = Zeiger auf Zielende

Ausgabe: –

Da das neue Modul das Programm verkleinern oder vergrößern kann, werden jetzt die Variablentabellen zwischen LOMEM und STREND bis unter FRETOP geschoben (wieder mit BLTU2) und das untere Ende vermerkt. Bis dorthin ist im äußersten Fall Platz für das Overlay. Die alten Zeiger auf der Zero-Page werden in das obere Ende des Tastaturpuffers gerettet.
Jetzt ist es Zeit, den neuen File zu lesen. Wir benutzen dazu einige Routinen von DOS 3.3 direkt. Da diese nicht von Apple dokumentiert sind, besteht keine Gewähr, daß sie bei Derivaten wie z.B. Diversi-DOS an der selben Stelle liegen. Falls das Programm bei Ihnen einmal nicht läuft, versuchen Sie es vor einer Fehlersuche erst mit normalem DOS 3.3.
Bevor nun der File gelesen wird, stellen wir vorsichtshalber seine Länge fest und vergleichen diese mit dem freien Platz im Speicher. Ist zuwenig Platz, gibt es eine Fehlermeldung. Die Länge des Files finden wir bei einem Applesoft-Programm in den ersten beiden Bytes des Files. Um diese zu lesen, müssen wir den File öffnen. Das ist etwas anderes als ein „OPEN" für Textfiles, da hierbei lediglich die Filepuffer und -Zeiger eingerichtet werden.
Damit wir wissen, welchen File wir öffnen sollen, müssen wir seinen Namen einlesen. FRMEVL ($DD7B) besorgt dies. Da FRMEVL aber nicht nur Zeichenketten auswertet, folgt mit CHKSTR ($DD6C) ein Test, ob wirklich eine Zeichenkette gelesen wurde. Ist das der Fall, wurde in $00A0 bis $00A2 ein vorläufiger Deskriptor angelegt. Wir benutzen diese Information, um den Filenamen in den Filenamens-Puffer des DOS zu kopieren, nachdem der Puffer vorher mit CLRFNBUF ($A095) mit Leerzeichen aufgefüllt worden ist. Für die weitere Bearbeitung muß Bit 7 gesetzt sein, was wir beim Kopieren mittels ORA #$80 gleich miterledigen. Der vorläufige Deskriptor wird danach mit FRESTR ($E5FD) wieder freigesetzt, damit er nicht auf dem Deskriptoren-Stack verbleibt.

FRESTR $E5FD
Entfernt einen String aus dem Stringpool (falls dort zuunterst) und löscht gegebenenfalls den Eintrag im Deskriptoren-Stack.

Eingabe:
VALTYP ($0011) muß $FF sein

Ausgabe:
Akku = Stringlänge
X-Reg/Y-Reg = Zeiger auf String (Lo/Hi)

Da DOS jetzt direkt aufgerufen werden soll, wird DOS aus dem Datenverkehr über die CSW- und KSW-Vektoren ausgeschlossen.

SETKBD $FE89
KSW wird auf $FD1B (KEYIN) gesetzt (= IN #0)

Eingabe: –

Ausgabe: KSWL/H ($0038/39) enthält $FD1B

SETVID $FE93
CSW wird auf $FDF0 (COUT1) gesetzt (= PR #0)

Eingabe: –

Ausgabe: CSWL/H ($0036/37) enthält $FDF0

Für unsere weitere Bearbeitung des Files benutzen wir einen Teil des DOS, der File-Manager genannt wird. Dieser kann durch einen Vektor auf der Seite 3 mittels JSR $03D6 aufgerufen werden. Damit der File-Manager weiß, was er tun soll, muß vorher eine Parameterliste aufgefüllt werden. Die Anfangsadresse dieser Liste finden wir durch ein JSR $03DC. Y-Register (Lo) und Akkumulator (Hi) bringen die Anfangsadresse zurück, die wir dann z.B. auf der Zero-Page speichern können, um sie für die indirekte indizierte Adressierung zu benutzen, denn alle Angaben in der Parameterliste erfolgen relativ zu ihrem Beginn. In unserem „Applesoft-Overlay-Manager" habe ich etwas unsauber programmiert und gleich die Standardadresse $B5BB für normales 48K-DOS benutzt, da auch die anderen Direktaufrufe nur mit diesem DOS funktionieren (nicht mit verschobenem DOS!).
ZEROPRMS ($A1AE) löscht die Parameterlister des File-Managers. Anschließend müssen wir einen freien File-Puffer suchen. GETBUF ($A764) erledigt dies und schreibt die Adresse nach NXTBUF ($0044/45). Ist das Hi-Byte gleich Null, ist kein Puffer mehr frei und wir geben die Fehlermeldung „NO BUFFERS AVAILABEL" aus (NOBUFS $A6C8). Für den File-Manager kopieren wir die gefundene Pufferadresse nach $0040/41 (CURBUF). COPYFN ($A743) kopiert den Filenamen aus dem Filenamenspuffer des File-Managers in den

gefundenen File-Puffer und deklariert diesen damit als belegt. COPYBP ($A74E) überträgt aus dem File-Puffer einige Zeiger in die Parameterliste. Dies sind: Adresse des File-Manager-Arbeitsbereich, Adresse des TSL-Puffers (für Track-Sektor-Liste des Files), Adresse des Datenpuffers, Adresse des nächsten File-Puffers. Wir brauchen uns um die Einrichtung dieser Puffer nicht zu kümmern. CMPLPRMS ($A71A) vervollständigt die Parameterliste weiter.

Jetzt kommt der spannende Moment. Das erste Byte der Parameterliste enthält das Befehlsbyte. Für „OPEN" muß dort $01 stehen. Daß wir das X-Register für dieses Laden benutzen, hat noch einen weiteren Effekt. Wenn der aufgerufene File noch nicht existiert, legt der File-Manager einen File an, wenn das X-Register beim Aufruf gleich Null ist. Da wir dies hier nicht wollen, verhindert das Laden des X-Registers mit $01 gleichzeitig auch die Neuanlage eines Files. Ist nach dem JSR FMGR das Carry-Bit gelöscht, war die Aktion erfolgreich, und wir können weitermachen. Im anderen Fall finden wir im zehnten ($0A) Byte der Parameterliste die Fehlernummer, die wir dazu benutzen, über DOSERR ($A6D2) eine Fehlermeldung auszugeben.
Im siebten Byte der Parameterliste finden wir den Filetyp des geöffneten Files. Bei Applesoft muß es $02 oder $82 bei einem „gelockten" File sein. Der Fehler $0D bedeutet „FILE TYPE MISMATCH". Mehr darüber in Kapitel 9.1.

Mit Hilfe von RD2BYTE ($A47A) lesen wir die ersten beiden Bytes des geöffneten Files, die ja die Filelänge enthalten. Reicht der Platz nicht, wird eine Meldung ausgegeben. Zumeist wird aber genügend RAM frei sein, und wir können über FILEREAD ($A471) den gesamten Rest des Files hereinholen. Damit der File-Manager auch weiß, wohin alles gelesen werden soll, müssen wir in den Indexregistern (X-Reg = Lo, Y-Reg = Hi) die gewünschte Ladeadresse übergeben. FILEREAD ist auch so freundlich und schließt automatisch den File für uns, so daß wir die Puffer nicht selber freigeben müssen. Mit CONNECT ($03EA) hängen wir DOS 3.3 wieder an, indem die CSW- und KSW-Vektoren zurückgesetzt werden.

Beim Lesen von DATA-Zeilen merkt sich Applesoft, welches Element zuletzt gelesen wurde. Wenn der DATA-Zeiger (DATPTR $007D/7E) in das Overlay-Gebiet zeigt, ist er jetzt nicht mehr gültig. Wir führen deshalb zwangsweise ein RESTORE ($D849) durch.

RESTORE $D849
Setzt den DATA-Zeiger auf den Programmanfang

Eingabe: –

Ausgabe: DATPTR ($007D/7E) = TXTTAB – 1

Wir sind jetzt fast fertig. Der neu hinzugeladene Teil muß nur noch mit dem alten Teil verknüpft werden, indem alle Zeiger auf die Zeilenanfänge neu berechnet werden. Die ROM-Routine von Applesoft können wir diesmal nicht benutzen, da sie nicht zurückkehrt. Wir bilden sie deshalb modifiziert in unserem Programm nach. STKINI ($D683) setzt den Stackzeiger auf $F8. Dadurch werden alle RETURN-Adressen gelöscht. Die Folge ist, daß bei einem Aufruf des „Applesoft-Overlay-Managers" keine offenen GOSUBs mehr bestehen dürfen.

STKINI $D683
Initiallisiert Stackzeiger

Eingabe: –

Ausgabe: Stackzeiger = $F8

Das neue Setzen vom LOMEM haben wir bereits besprochen. Die Variablentabellen, die bis jetzt unter FRETOP hingen, werden wieder bis zum neuen LOMEM-Wert nach unten geschoben. Für diese Verschiebung nach unten benutzen wir die Routine MOVE ($FE2C). Interessant ist ferner die Neuberechnung der diversen Zeiger, die im Tastaturpuffer zwischengespeichert waren.

MOVE $FE2C
Verschiebt den Speicherbereich zwischen A1L/H ($003C/3D) und A2L/H ($003E/3F) zur Zieladresse A4L/H ($0042/43)
Eignet sich nicht zur Verschiebung nach oben bei Bereichsüberschneidung.

Eingabe:
A1L/H ($003C/3D) = Startadresse (Lo/Hi)
A2L/H ($003E/3F) = Endadresse (Lo/Hi)
A4L/H ($0042/43) = Zieladresse (Lo/Hi)
Y-Reg = $00 (wichtig!)

Ausgabe:
Carry gesetzt

Eine allerletzte Zeile fehlt uns noch. NEWSTT ($D7D2) springt zurück ins Applesoft-Programm und führt den nächsten Befehl nach unserem Ampersand-Aufruf durch.

NEWSTT $D7D2
Führt nächsten Applesoft-Befehl aus.

Eingabe:
TXTPTR zeigt auf das letzte Zeichen vor dem neuen Befehl („:“ oder EOL)

9. Blitz-Leser

Falls Sie schon einmal mit CP/M gearbeitet haben, werden Sie sicher den TYPE-Befehl kennen, mit dem Sie sich Textfiles ansehen können. Unter DOS besitzen wir derartige Möglichkeiten nicht. Wir benötigen zum Lesen von sequentiellen Textfiles ein eigenes Programm. Dafür werden wir es aber auch komfortabler gestalten als das TYPE.

Bedienungsanleitungen zu Programmen sind oft als Textfiles abgespeichert. Auch manche Textverarbeitungsprogramme erzeugen reine Textfiles. Unser schneller File-Leser wird diese Texte wie der Blitz von der Diskette lesen und auf den 40Z/Z-Bildschirm ausgeben. Mit der Rechts- und der Linkspfeil-Taste können Sie die Geschwindigkeit der Ausgabe steuern, um sie Ihren Lesegewohnheiten anzupassen. An beiden Enden der Verstellmöglichkeit macht Sie ein Ton darauf aufmerksam, daß Sie das Maximum erreicht haben. Die Leertaste stoppt die Ausgabe zeitweise, mit <ESC> können Sie sie abbrechen. Ein Drücken von <RETURN> springt zur schnellstmöglichen Ausgabe.

Die Idee zu diesem Programm kam mir beim „File-Reader" von Ulrich Stiehl (Apple DOS 3.3, Seite 186 der 2. Auflage). Ich habe dieses Programm häufig benutzt, und Sie werden bei der Bildschirmausgabe einige Ähnlichkeiten entdecken können. Zwei Dinge haben mich gestört:
1) Die Geschwindigkeit ist nicht gut regulierbar.
Hier ist die Abhilfe leicht möglich.
2) Die Diskette mit dem Textfile darf nicht schreibgeschützt sein.
Dieses Problem ist schon schwieriger. Ulrich Stiehl wendet den von ihm erdachten „UNLOCK"-Trick an, um auf einfache Weise in den Besitz der Track-Sektor-Liste (TSL) des Files zu kommen. UNLOCK schreibt aber auf die Diskette, auch wenn der File nicht gesperrt sein sollte.
Wenn wir den schon aus dem letzten Kapitel bekannten File-Manager einsetzen, können wir mit relativ geringem Mehraufwand auch an die TSL herankommen, ohne daß dazu ein Schreibzugriff nötig wäre. Das Kommando „OPEN" des

File-Managers (Achtung, nicht identisch mit dem DOS-Befehl OPEN!) liest ebenfalls die erste TSL des eröffneten Files ein.
Die weitere Arbeit lassen wir durch einen zweiten Programmteil des DOS machen. Die RWTS (Read/Write Track/Sektor) liest ganze Sektoren in den Speicher. Dies geschieht über die RWTS bis zu 15-mal schneller als mit normalen DOS 3.3-Kommandos, die viel Zeit für Verwaltungsaufgaben verlieren. Nebenbei bemerkt: ProDOS besitzt schnellere Schreib- und Leseroutinen (RWTB = Read/Write Track/Block) als DOS, betreibt aber einen noch größeren Verwaltungsaufwand, so daß diese Vorteile zum Teil wieder verloren gehen.
Wie für den File-Manager müssen wir auch hier eine Parameterliste anlegen und Vektoren auf der Seite 3 benutzen. Bevor wir dies im einzelnen besprechen, können Sie sich jetzt schon in das Listing vertiefen.

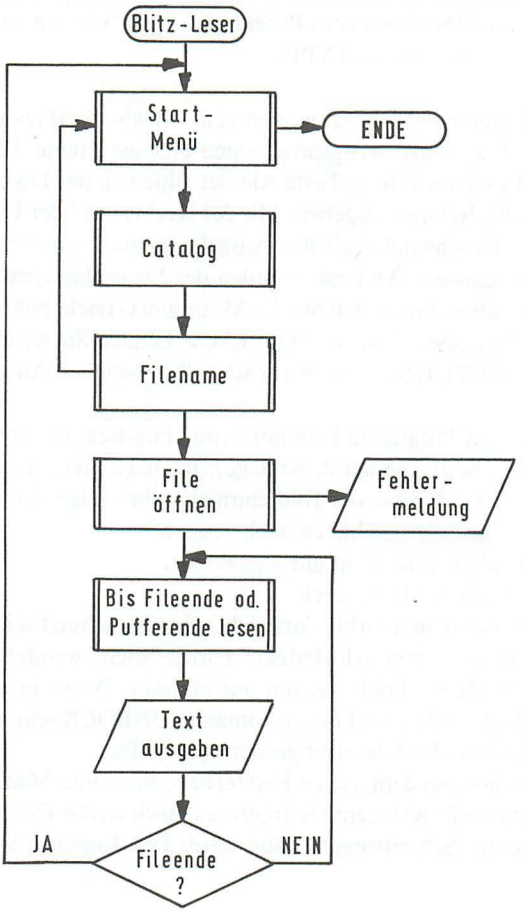


Abb. 11: Vereinfachter Ablaufplan des schnellen File-Lesers

FILELESER

```
                    1   ;*******************************
                    2   ;   Schneller File-Leser für   *
                    3   ;          Text-Files           *
                    4   ; (C) 1986 Dr. Jürgen Kehrel    *
                    5   ;*******************************
                    6   ;         Version 2.0
                    7   ;
                    8   PTRIOB    EQU $0000      ;+$0001
                    9   PTRFMPL   EQU $0002      ;+$0003
                   10   PTRTSL    EQU $0004      ;+$0005
                   11   PTRTRK    EQU $0006
                   12   PTRPUF    EQU $0007      ;+$0008
                   13   YSAVE     EQU $0009
                   14   CH        EQU $0024      ;Cursor horz.
                   15   PROMPT    EQU $0033      ;Prompt
                   16   PREG      EQU $0048      ;P-Register
                   17   PTRPRT    EQU $00CE      ;+$00CF
                   18   ;
                   19   IN        EQU $0200      ;Tastaturpuf.
                   20   ;
                   21   DOSCLD    EQU $03D3
                   22   FLEMGR    EQU $03D6
                   23   RWTS      EQU $03D9
                   24   GETFMPL   EQU $03DC
                   25   GETIOB    EQU $03E3
                   26   CONNECT   EQU $03EA
                   27   ;
                   28   PUFFER    EQU $1000      ;$1000 - $8AFF
                   29   ;
                   30   KBD       EQU $C000
                   31   STROBE    EQU $C010
                   32   ;
                   33   TEXT      EQU $FB2F
                   34   TABV      EQU $FB5B      ;Position
                   35   BELL      EQU $FBDD
                   36   CLREOP    EQU $FC42      ;Löschen
                   37   HOME      EQU $FC58
                   38   WAIT      EQU $FCA8      ;Warteschleife
                   39   RDKEY     EQU $FD0C      ;Taste lesen
                   40   GETLN     EQU $FD6A      ;Zeile lesen
                   41   SETKBD    EQU $FE89
                   42   SETVID    EQU $FE93
                   43   COUT      EQU $FDED
                   44   ;
                   45   ;
                   46             ORG $803
                   47   ;
                   48   ;
                   49   ; Titel und Menü ausgeben
                   50   ;
0803: D8           51   START     CLD            ;Binärmode
0804: 20 2F FB     52             JSR TEXT
0807: 20 58 FC     53             JSR HOME
080A: A9 07        54             LDA #7
```

```
080C: 85 24     55            STA CH
080E: A9 0A     56            LDA #10
0810: 20 5B FB  57            JSR TABV
0813: 20 3E 0B  58            JSR PRINT
0816: AA A0 D3  59            ASC "* SCHNELLER FILE-LESER *"
082E: 00        60            HEX 00
082F: A9 06     61            LDA #6
0831: 85 24     62            STA CH
0833: A9 14     63            LDA #20
0835: 20 5B FB  64            JSR TABV
0838: 20 3E 0B  65            JSR PRINT
083B: D6 CF CE  66            ASC "VON  DR. JUERGEN"
084B: A0 C2 AE  67            ASC " B. KEHREL"
0855: 8D 00     68            HEX 8D,00
0857: A9 10     69            LDA #16
0859: 85 24     70            STA CH
085B: 20 3E 0B  71            JSR PRINT
085E: A8 B1 B9  72            ASC "(1986)"
0864: 8D 00     73            HEX 8D00
0866: A2 32     74  MENUE     LDX #$32
0868: A9 C4     75  WARTEN    LDA #$C4
086A: 20 A8 FC  76            JSR WAIT
086D: AD 00 C0  77            LDA KBD
0870: 10 05     78            BPL WARTE1
0872: 8D 10 C0  79            STA STROBE
0875: 30 03     80            BMI MENUE1
0877: CA        81  WARTE1    DEX
0878: D0 EE     82            BNE WARTEN
                83  ;
087A: 20 58 FC  84  MENUE1    JSR HOME
087D: A9 0A     85            LDA #10
087F: 20 5B FB  86            JSR TABV
0882: 20 3E 0B  87            JSR PRINT
0885: CC C5 C7  88            ASC "LEGEN SIE ..."
0892: 8D 8D     89            HEX 8D8D
0894: C4 C9 C5  90            ASC "DIE DISKETTE MIT DEM TEXTFILE"
08B1: 8D 8D     91            HEX 8D8D
08B3: A0 A0 A0  92            ASC "       IN LAUFWERK 1"
08C7: 8D 8D 8D  93            HEX 8D8D8D8D
08CB: BC D2 C5  94            ASC "‹RETURN› = START,"
08DC: A0 BC C5  95            ASC " ‹ESC› = ENDE "
08EA: 00        96            HEX 00
08EB: 20 0C FD  97  EINGABE   JSR RDKEY
08EE: C9 8D     98            CMP #$8D        ;Return
08F0: F0 0A     99            BEQ DIR
08F2: C9 9B     100           CMP #$9B        ;Escape
08F4: D0 F5     101           BNE EINGABE
08F6: 20 58 FC  102           JSR HOME
08F9: 4C D3 03  103           JMP DOSCLD
                104 ;
                105 ; Lies DOS-Catalog
                106 ;
08FC: 20 EA 03  107 DIR       JSR CONNECT     ;DOS anhängen
08FF: 20 3E 0B  108           JSR PRINT
```

```
0902: A0 8D 84  109           HEX A08D84
0905: C3 C1 D4  110           ASC "CATALOG,D1"
090F: 8D 00     111           HEX 8D00
0911: 20 0C FD  112           JSR RDKEY
0914: 4C 31 09  113           JMP FERTIG      ;immer
                114 ;
                115 ; Textfilename (Eingabefile)
                116 ;
0917: 20 42 FC  117 ZULANG    JSR CLREOP      ;Lösche bis
091A: 20 3E 0B  118           JSR PRINT       ;Seitenende
091D: 87 87 8D  119           HEX 87878D      ;Piep
0920: 05 09 0E  120           INV "EINGABE ZU LANG"
092F: 8D 00     121           HEX 8D00
                122 ;
0931: 20 3E 0B  123 FERTIG    JSR PRINT
0934: A0 8D 8D  124           HEX A08D8D00
0938: A9 05     125           LDA #5
093A: 85 24     126           STA CH
093C: 20 3E 0B  127           JSR PRINT
093F: CE C1 CD  128           ASC "NAME DES TEXTFILES ?"
0953: 8D A0 A0  129           HEX 8DA0A0A0A0A02D00
095B: A9 3E     130           LDA #$3E        ;Prompt-Zeichen
095D: 85 33     131           STA PROMPT
095F: 20 89 FE  132           JSR SETKBD      ;DOS abhängen
0962: 20 93 FE  133           JSR SETVID
0965: 20 6A FD  134           JSR GETLN       ;Eingabe von
0968: E0 1F     135           CPX #31         ;max. 30 Z
096A: B0 AB     136           BGE ZULANG
096C: 8A        137           TXA
096D: D0 03     138           BNE LESEN       ;nur Return
096F: 4C 7A 08  139           JMP MENUE1      ;zurück
                140 ;
                141 ;
                142 ; Tastaturpuffer -› Filenamenpuffer
                143 ;
0972: A0 1D     144 LESEN     LDY #29         ;Filenamenpuffer
0974: A9 A0     145           LDA #$A0        ;löschen
0976: 99 78 0B  146 LOESCH    STA FILE,Y
0979: 88        147           DEY
097A: 10 FA     148           BPL LOESCH
                149 ;
097C: CA        150           DEX             ;ohne ‹cr›
097D: BD 00 02  151 COPY      LDA IN,X        ;Tastaturpuffer
0980: 09 80     152           ORA #$80        ;Bit7 setzen
0982: 9D 78 0B  153           STA FILE,X      ;Namenpuffer
0985: CA        154           DEX             ;alles über-
0986: 10 F5     155           BPL COPY        ;tragen ?
                156 ;
                157 ;
                158 ; Alles in Ordnung? Sonst zurück
                159 ;
0988: 8E F3 0A  160           STX VERZUG+1    ;auf Maximum setzen
098B: 20 EA 03  161           JSR CONNECT     ;DOS anhängen
098E: 20 42 FC  162           JSR CLREOP      ;Löschen
```

```
0991: 20 3E 0B  163            JSR PRINT
0994: 8D 8D     164            HEX 8D8D
0996: BC D2 C5  165            ASC "‹RETURN› = FERTIG,"
09A8: A0 BC C5  166            ASC " ‹ESC› = ZURUECK "
09B9: 00        167            HEX 00
09BA: 20 0C FD  168 READKEY    JSR RDKEY       ;Tastendruck
09BD: C9 8D     169            CMP #$8D        ;Return
09BF: F0 07     170            BEQ RETURNS
09C1: C9 9B     171            CMP #$9B        ;Escape
09C3: D0 F5     172            BNE READKEY
09C5: 4C 7A 08  173            JMP MENUE1
                174 ;
09C8: 20 ED FD  175 RETURNS    JSR COUT        ;3* ‹CR› ausgeben
09CB: 20 ED FD  176            JSR COUT
09CE: 20 ED FD  177            JSR COUT
                178 ;
                179 ;
                180 ; Adressen des I/O-Blocks der RWTS
                181 ; und der Parameterliste des File-
                182 ; managers holen und eintragen.
                183 ;
09D1: 20 E3 03  184 RDFILE     JSR GETIOB
09D4: 84 00     185            STY PTRIOB
09D6: 85 01     186            STA PTRIOB+1
                187 ;
09D8: 20 DC 03  188            JSR GETFMPL
09DB: 84 02     189            STY PTRFMPL
09DD: 85 03     190            STA PTRFMPL+1
                191 ;
                192 ; File mit dem Filemanager öffnen.
                193 ; Parameterliste dafür vorbereiten.
                194 ; Der zuletzt benutzte DOS-Puffer
                195 ; (vom CATALOG) wird erneut benutzt.
                196 ;
09DF: A9 01     197            LDA #$01        ;„OPEN"
09E1: A0 00     198            LDY #$00        ;Typ des Aufrufs
09E3: 91 02     199            STA (PTRFMPL),Y
09E5: A9 00     200            LDA #$00        ;„Textdatei"
09E7: A0 07     201            LDY #$07        ;Typ des Files
09E9: 91 02     202            STA (PTRFMPL),Y
09EB: C8        203            INY             ;Adresse Filename LO
09EC: A9 78     204            LDA #‹FILE
09EE: 91 02     205            STA (PTRFMPL),Y
09F0: C8        206            INY             ;Adresse Filename HI
09F1: A9 0B     207            LDA #›FILE
09F3: 91 02     208            STA (PTRFMPL),Y
                209 ;
09F5: A0 01     210            LDY #$01        ;SLOT *16
09F7: B1 00     211            LDA (PTRIOB),Y
09F9: 4A        212            LSR
09FA: 4A        213            LSR
09FB: 4A        214            LSR
09FC: 4A        215            LSR             ;:16
09FD: A0 06     216            LDY #$06        ;SLOT
```

```
09FF: 91 02     217             STA (PTRFMPL),Y
0A01: A0 02     218             LDY #$02        ;DRIVE
0A03: B1 00     219             LDA (PTRIOB),Y
0A05: A0 05     220             LDY #$05        ;DRIVE
0A07: 91 02     221             STA (PTRFMPL),Y
0A09: A2 01     222             LDX #$01        ;nicht neu einrichten
0A0B: 20 D6 03  223             JSR FLEMGR      ;File öffnen
0A0E: 90 1D     224             BCC READ_IT     ;kein Fehler
                225 ;
                226 ; Mögliche Fehler sind „FILE
                227 ; NOT FOUND" und „I/O ERROR"
                228 ;
0A10: A0 0A     229             LDY #$0A        ;Fehler-Code
0A12: B1 02     230             LDA (PTRFMPL),Y
0A14: C9 06     231             CMP #$06        ;FILE NOT FOUND
0A16: F0 02     232             BEQ ERRLOOP
0A18: A0 00     233 IOERR       LDY #$00        ;sonst I/O ERROR
0A1A: B9 5F 0B  234 ERRLOOP     LDA ERRTEXT,Y   ;Texte ausgeben
0A1D: F0 06     235             BEQ TOMENUE
0A1F: 20 ED FD  236             JSR COUT
0A22: C8        237             INY
0A23: D0 F5     238             BNE ERRLOOP
                239 ;
0A25: 20 DD FB  240 TOMENUE     JSR BELL
0A28: 84 48     241             STY PREG        ;auf Null setzen
0A2A: 4C 66 08  242             JMP MENUE       ;Pause, dann Menü
                243 ;
                244 ; File erfolgreich geöffnet, jetzt
                245 ; die einzelnen Datensektoren di-
                246 ; rekt über RWTS in den Puffer lesen.
                247 ;
0A2D: A0 0E     248 READ_IT     LDY #$0E        ;TSL-Adresse LO
0A2F: B1 02     249             LDA (PTRFMPL),Y
0A31: 85 04     250             STA PTRTSL
0A33: C8        251             INY             ;TSL-Adresse HI
0A34: B1 02     252             LDA (PTRFMPL),Y
0A36: 85 05     253             STA PTRTSL+1
0A38: A9 0C     254             LDA #$0C        ;Offset in TSL-Liste
0A3A: 85 06     255             STA PTRTRK      ;zum ersten Eintrag
0A3C: A0 08     256             LDY #$08        ;Lesepuffer LO
0A3E: A9 00     257             LDA #<PUFFER
0A40: 91 00     258             STA (PTRIOB),Y
0A42: C8        259             INY             ;Lesepuffer HI
0A43: A9 10     260             LDA #>PUFFER
0A45: 91 00     261             STA (PTRIOB),Y
0A47: A9 00     262             LDA #$00
0A49: A0 03     263             LDY #$03        ;VOLUME
0A4B: 91 00     264             STA (PTRIOB),Y
0A4D: A0 0C     265             LDY #$0C        ;Kommando
0A4F: A9 01     266             LDA #$01        ;„READ"
0A51: 91 00     267             STA (PTRIOB),Y
                268 ;
                269 ; Sektor für Sektor die TSL abarbeiten
                270 ;
```

```
0A53: A4 06     271  READLOOP  LDY PTRTRK
0A55: B1 04     272            LDA (PTRTSL),Y ;Track
0A57: F0 24     273            BEQ READEND    ;kein weiterer Sektor
0A59: A0 04     274            LDY #$04       ;Track
0A5B: 91 00     275            STA (PTRIOB),Y
0A5D: A4 06     276            LDY PTRTRK
0A5F: C8        277            INY            ;Sektor
0A60: B1 04     278            LDA (PTRTSL),Y
0A62: A0 05     279            LDY #$05       ;Sektor
0A64: 91 00     280            STA (PTRIOB),Y
0A66: 20 E3 03  281            JSR GETIOB
0A69: 20 D9 03  282            JSR RWTS
0A6C: B0 AA     283            BCS IOERR      ;RWTS-Fehler
                284  ;
0A6E: A0 09     285            LDY #$09       ;Puffer Hi-Byte
0A70: B1 00     286            LDA (PTRIOB),Y
0A72: 18        287            CLC
0A73: 69 01     288            ADC #$01       ;+1
0A75: 91 00     289            STA (PTRIOB),Y
0A77: E6 06     290            INC PTRTRK     ;Zeiger in der TSL
0A79: E6 06     291            INC PTRTRK     ;weitersetzen
0A7B: D0 D6     292            BNE READLOOP
                293  ;
                294  ; Die eingelesenen Sektoren anzeigen
                295  ;
                296  ;
0A7D: A0 09     297  READEND   LDY #$09       ;Puffer Hi
0A7F: B1 00     298            LDA (PTRIOB),Y
0A81: 8D 01 0B  299            STA TESTEND+1  ;Endwert „poken"
                300  ;
0A84: A9 00     301            LDA #‹PUFFER
0A86: 85 07     302            STA PTRPUF
0A88: A9 10     303            LDA #›PUFFER
0A8A: 85 08     304            STA PTRPUF+1
                305  ;
0A8C: A0 00     306            LDY #0
0A8E: B1 07     307  AUSGABE1  LDA (PTRPUF),Y
0A90: F0 76     308            BEQ ZUENDE     ;File zu Ende
0A92: 09 80     309            ORA #$80       ;Bit7 setzen
0A94: C9 8D     310            CMP #$8D       ;‹cr›?
0A96: F0 06     311            BEQ AUSGABE2   ;Ja, ausgeben
0A98: C9 A0     312            CMP #$A0       ;Controll-Zeichen?
0A9A: B0 02     313            BGE AUSGABE2   ;Nein, ausgeben
0A9C: 29 1F     314            AND #$1F       ;%0001 1111 (Inverse)
0A9E: 20 ED FD  315  AUSGABE2  JSR COUT
0AA1: AD 00 C0  316            LDA KBD        ;Tastendruck?
0AA4: 10 4C     317            BPL VERZUG     ;Nein, weiter
                318  ;
0AA6: 2C 10 C0  319            BIT STROBE     ;Ja, zurücksetzen
0AA9: C9 9B     320            CMP #$9B       ;Escape ?
0AAB: D0 03     321            BNE LEER
0AAD: 4C 37 0B  322            JMP AUSGABE4   ;Abbruch
0AB0: C9 A0     323  LEER      CMP #$A0       ;Leertaste ?
0AB2: D0 0A     324            BNE CR         ;andere testen
```

```
0AB4: AD 00 C0   325 PAUSE    LDA KBD        ;Pause bis Tastendruck
0AB7: 10 FB      326           BPL PAUSE
0AB9: 8D 10 C0   327           STA STROBE     ;zurücksetzen
0ABC: 30 3B      328           BMI AUSGABE3   ;direkt weiter
                 329 ;
0ABE: C9 8D      330 CR        CMP #$8D       ;‹CR›
0AC0: D0 07      331           BNE PFEIL
0AC2: A9 00      332           LDA #$00       ;max. Geschwindigkeit
0AC4: 8D F3 0A   333           STA VERZUG+1   ;„poken"
0AC7: F0 30      334           BEQ AUSGABE3   ;immer weiter
                 335 ;
0AC9: C9 88      336 PFEIL     CMP #$88       ;‹-
0ACB: D0 15      337           BNE SCHNELL
0ACD: AD F3 0A   338           LDA VERZUG+1   ;Verzögerungswert
0AD0: C9 FB      339           CMP #$FB
0AD2: B0 04      340           BCS PIEPS
0AD4: 69 05      341           ADC #$5        ;+5
0AD6: D0 17      342           BNE POKE
                 343 ;
0AD8: 84 09      344 PIEPS     STY YSAVE
0ADA: 20 DD FB   345           JSR BELL
0ADD: A4 09      346           LDY YSAVE
0ADF: 4C F9 0A   347           JMP AUSGABE3
                 348 ;
0AE2: C9 95      349 SCHNELL   CMP #$95       ;-›
0AE4: D0 0C      350           BNE VERZUG     ;keine Aktion
0AE6: AD F3 0A   351           LDA VERZUG+1   ;Verzögerung
0AE9: C9 05      352           CMP #$5        ;wenigstens noch
0AEB: 90 EB      353           BCC PIEPS
0AED: E9 05      354           SBC #$5        ;-5
0AEF: 8D F3 0A   355 POKE      STA VERZUG+1
                 356 ;
0AF2: A9 FF      357 VERZUG    LDA #$FF
0AF4: F0 03      358           BEQ AUSGABE3
0AF6: 20 A8 FC   359           JSR WAIT
0AF9: C8         360 AUSGABE3  INY            ;nächstes Zeichen
0AFA: D0 92      361           BNE AUSGABE1
                 362 ;
0AFC: E6 08      363           INC PTRPUF+1   ;Speicherseite weiter-
0AFE: A5 08      364           LDA PTRPUF+1   ;schalten, das Ende testen
0B00: C9 FF      365 TESTEND   CMP #$FF       ;Dummy-Wert (wird „gepokt")
0B02: 90 8A      366           BLT AUSGABE1   ;noch nicht zu Ende
                 367 ;
                 368 ; Prüfen, ob weitere TSL's da sind
                 369 ;
0B04: A0 01      370           LDY #$01       ;Link-Byte Track
0B06: B1 04      371           LDA (PTRTSL),Y
0B08: F0 25      372 ZUENDE    BEQ FILEEND    ;NEIN
0B0A: A0 04      373           LDY #$04       ;Track
0B0C: 91 00      374           STA (PTRIOB),Y
0B0E: A0 02      375           LDY #$02       ;Link-Byte Sektor
0B10: B1 04      376           LDA (PTRTSL),Y
0B12: A0 05      377           LDY #$05       ;Sektor
0B14: 91 00      378           STA (PTRIOB),Y
```

```
0B16: A5 04     379           LDA PTRTSL      ;Zieladresse
0B18: A0 08     380           LDY #$08        ;Puffer Lo
0B1A: 91 00     381           STA (PTRIOB),Y
0B1C: A5 05     382           LDA PTRTSL+1
0B1E: C8        383           INY             ;Puffer Hi
0B1F: 91 00     384           STA (PTRIOB),Y
0B21: 20 E3 03  385           JSR GETIOB
0B24: 20 D9 03  386           JSR RWTS
0B27: B0 03     387           BCS TOIOERR
0B29: 4C 2D 0A  388           JMP READ_IT
0B2C: 4C 18 0A  389 TOIOERR   JMP IOERR
                390 ;
0B2F: AD 00 C0  391 FILEEND   LDA KBD
0B32: 10 FB     392           BPL FILEEND
0B34: 2C 10 C0  393           BIT STROBE
0B37: A9 00     394 AUSGABE4  LDA #$00
0B39: 85 48     395           STA PREG        ;auf Null setzen
0B3B: 4C 7A 08  396           JMP MENUE1      ;zurück
                397 ;
                398 ; Nach Andy Hertzfeld Call Apple 8/81
                399 ;
0B3E: 68        400 PRINT     PLA             ;hole Return-
0B3F: 85 CE     401           STA PTRPRT      ;adresse vom
0B41: 68        402           PLA             ;Stack
0B42: 85 CF     403           STA PTRPRT+1
0B44: A0 01     404           LDY #1
0B46: B1 CE     405 PR1       LDA (PTRPRT),Y ;String
0B48: F0 06     406           BEQ ENDE
0B4A: 20 ED FD  407           JSR COUT        ;ausgeben
0B4D: C8        408           INY
0B4E: D0 F6     409           BNE PR1
0B50: 18        410 ENDE      CLC
0B51: 98        411           TYA             ;Stringlänge
0B52: 65 CE     412           ADC PTRPRT      ;addieren u.
0B54: 85 CE     413           STA PTRPRT
0B56: A5 CF     414           LDA PTRPRT+1
0B58: 69 00     415           ADC #0
0B5A: 48        416           PHA             ;auf Stack
0B5B: A5 CE     417           LDA PTRPRT      ;zurück,
0B5D: 48        418           PHA             ;dann dorthin
0B5E: 60        419           RTS             ;springen
                420 ;
                421 ; Fehlermeldungen und Puffer für Filenamen
                422 ;
0B5F: 09 2F 0F  423 ERRTEXT   INV "I/O ERROR"
0B68: 00        424           HEX 00
0B69: 06 09 0C  425           INV "FILE NOT FOUND"
0B77: 00        426           HEX 00
0B78: A0 A0 A0  427 FILE      ASC "                "
0B96: 00        428           HEX 00
```

Da der schnelle File-Leser ein eigenständiges Programm ist, können wir ihn an den Beginn des Hauptspeichers legen. Hier wäre ab $0800 Platz, aber wenn wir das Byte $0800 im Anschluß an den File-Leser nicht wieder auf $00 setzen würden, könnte ein BASIC-Programm nicht ordnungsgemäß laufen. Wenn wir erst bei $0803 beginnen, haben wir solche Probleme nicht.

Die ersten hundert Zeilen dürften Ihnen jetzt keine Mühe mehr machen. Es wird ein Titel gedruckt und ein kleines Menü gestartet. Der CATALOG der einliegenden Diskette wird erzeugt, indem wir die Zeichenkette <RETURN>,<CTRL-D>,CATALOG über COUT ausgeben. Da DOS in den Datenverkehr einbezogen ist (JSR CONNECT), wird der Befehl ausgeführt. Wenn sie genau hinschauen, wird Ihnen hier und in Zeile 124 noch ein A0 vor dem 8D des Returns auffallen. Hiermit hat es eine besondere Bewandtnis. Wenn wir bei aktivem DOS die Monitor-Routine RDKEY ($FD0C) benutzen und anschließend ein $8D (= Return) über COUT ausgeben, passiert etwas Merkwürdiges. Das DOS versucht, den Inhalt des Tastaturpuffers als DOS-Befehl auszuführen, auch wenn ihm kein Ctrl-D voransteht. Meistens steht zu diesem Zeitpunkt auch gar kein Befehl im Tastaturpuffer, sodaß wir schlimmstenfalls abstürzen. Wird erst irgendein anderes Zeichen außer $84 und $8D über COUT ausgegeben, verhält sich DOS wieder ganz normal. Das Beste ist es, einfach ein Leerzeicheen ($A0) zu senden, da dies auf dem Bildschirm nicht sichtbar ist.

RDKEY $FD0C
Zeigt den blinkenden Cursor und holt ein Zeichen über den KSW-Vektor (normal KEYIN $FD1B = Tastatur)

Eingabe: –

Ausgabe: Akku = gelesenes Zeichen

Zum Einlesen des File-Namens benutzen wir wieder GETLN ($FD6A). Der Name wird anschließend umkopiert in einen speziellen Zwischenspeicher FILE, den wir zuvor mit Leerzeichen gelöscht haben. Während dieser Aktion wird DOS vorübergehend abgekoppelt (Zeile 132/133), damit wir auch File-Namen benutzen können, die gleichzeitig DOS-Befehle sind. Bis zur Zeile 180 passiert dann nichts Aufregendes mehr.

Mit JSR GETIOB ($03E3) erhalten wir in Y-Register und Akkumulator die Adresse der RWTS-Parameterliste, die hier auch I/O-Block genannt wird. JSR GETFMPL besorgt selbiges für den File-Manager. Das Öffnen des Files über den File-Manager kennen Sie schon aus dem vorigen Kapitel, nur daß wir hier einige Parameter mehr „von Hand" setzen. Auch das Suchen eines freien File-Puffers ersparen wir uns und nutzen kurzerhand den selben Puffer wieder, der gerade zuvor von CATALOG angelegt wurde.
In der Parameterliste des File-Mangers finden wir an der relativen Position $E/F die Lage der TSL im Speicher. Auch hierzu wird ein Zeiger auf der Zero-Page angelegt (PTRTRK), um die Track/Sektor-Paare der RWTS zu übergeben. Alle Sektoren werden in den Speicher gelesen, bis der File zu Ende ist.
Bei sehr großen Files können zwei Dinge auftreten. Zum einen kann der Puffer voll sein. Dann wird der Rest des Files in einem weiteren Schub eingelesen. Zum zweiten haben große Files mehr als eine TSL. In jeder TSL finden wir an den Positonen $01 und $02 einen Zeiger zur nächsten TSL. Erst wenn der Trackwert $00 ist, haben wir alle TSL's gefunden. Unser File wird so bis zum letzen Sektor eingelesen.

9.1 Eine kleine DOS-Enzyklopädie

Da wir in unseren zwei DOS-Programmen nicht auf alle Möglichkeiten der RWTS und des File-Managers eingehen konnten, werden jetzt tabellarisch die Besonderheiten dieser beiden Unterprogramme vorgestellt.

Die Parameter-Liste der RWTS (IOB) ist wie folgt aufgebaut:

```
Byte $00  01  Konstante
Byte $01  60  jetziger Slot * 16 (hier Slot 6)
Byte $02  01  jetziges Laufwerk (1 oder 2)
Byte $03  00  erwartete Volume-Nummer ($00 paßt für alles)
Byte $04  04  gewünschter Track ($00-$22, hier $04)
Byte $05  0F  gewünschter Sektor ($00-$0F, hier $0F)
Byte $06  Lo  Lo-Byte der DCT (s.u.)
Byte $07  Hi  Hi-Byte der DCT
Byte $08  Lo  Lo-Byte des Datenpuffers (256 Bytes)
Byte $09  Hi  Hi-Byte des Datenpuffers
Byte $0A  00  Konstante
Byte $0B  00  Konstante
Byte $0C  02  Befehlsbyte (s.u.)
Byte $0D  00  Fehlercode, wird vom DOS gepoked
Byte $0E  00  alte Volume-Nummer
```

```
Byte $0F  60  letzter benutzter Slot (hier Slot 6)
Byte $10  01  letztes benutztes Laufwerk
```

Alle Bytes verstehen sich relativ zum Beginn, dessen absolute Adresse Sie durch ein JSR GETIOB erhalten (Y-Reg = Lo, Akku = Hi).
Die Bytes $06 und $07 enthalten einen Zeiger auf eine weitere Tabelle, die DCT (Device Characteristics Table). Für ein DISK II-Laufwerk hat sie immer das folgende Aussehen:

```
Byte $00 ($11) 00 Konstante (Laufwerkstyp)
Byte $01 ($12) 01 Konstante (Phasen per Track - 1)
Byte $02 ($13) EF Konstante (Anlaufzähler)
Byte $03 ($14) D8 Konstante (Anlaufzähler)
```

Da die DCT zumeist direkt hinter dem IOB liegt, sind in Klammern die Bytes weitergezählt.
Das Byte $0C enthält einen der folgenden Befehle:
$00 Seek = nur Track/Sektor suchen (Kopfbewegung)
$01 Read = bezeichneten Sektor lesen
$02 Write = bezeichneten Sektor schreiben
$04 Init = Diskette **ganz** initialisieren (ohne HELLO-Programm)

Bevor Sie die Codes $02 und $04 ausprobieren, sollten Sie sicher sein, daß Ihr Programm auch funktioniert!
Wenn Sie den IOB des DOS mitbenutzen, müssen Sie die Bytes $06/07, $0D bis $10 sowie die DCT nicht selber initialisieren. Schreiben Sie dagegen einen eigenen IOB, müssen alle Bytes gesetzt werden.
Damit die RWTS weiß, welcher IOB benutzt werden soll, ist beim RWTS-Aufruf die IOB-Adresse in Y-Register (Lo) und Akkummulator (Hi) zu übergeben. Im Normalfall geschieht das so:

```
JSR GETIOB              LDY #<IOB   ;Lo-Byte
JSR RWTS                LDA #>IOB   ;Hi-Byte
                        JSR RWTS
DOS-Tabelle nutzen      eigene Tabelle nutzen
```

Wenn nach dem JSR RWTS das Carry-Bit gesetzt ist, trat ein Fehler auf, dessen Code im relativen Byte $0D zu finden ist. Die RWTS verwendet intern die Speicherstelle $0048, die gleichzeitig auch vom Monitor benutzt wird. Damit es hier nicht zu Konflikten kommt, können wir $0048 nach jedem JSR RWTS auf $00 setzen.

Der File-Manager besitzt noch mehr Befehle als die RWTS. Folglich fällt auch die Parameterliste komplizierter aus. Sie müssen allerdings nicht immer alle

Werte initialisieren. Deshalb sind in der Tabelle nur die notwendigen Bytes eingetragen. Die gesamten Pufferadressen können Sie auch von DOS eintragen lassen, wenn Sie mit GETBUF ($A764) einen Puffer suchen und anschließend mit COPYFN, COPYBP und CMPLPRMS die Parameter übertragen. Ein Beispiel hatten wir in Kapitel 8. Der Nachteil ist, daß Sie hier absolute Adressen benutzen müssen, die nur für ein nichtverschobenes 48K-DOS gelten. Wenn Ihr Programm unter allen Umständen laufen soll, kommen Sie nicht umhin, alle Parameter selber zu setzen.

Befehl:	OPEN	CLOSE	READ	WRITE	DEL.	CAT.	LOCK	UNL.	REN.	POS.	INIT	VERIFY
Byte $00	01	02	03	04	05	06	07	08	09	0A	0B	0C
Byte $01	--	--	Untergruppe		--	--	--	--	--	--	DOS-P	-
Byte $02	Rec.-	--	Record-		--	--	--	--	neuer	Rec.	--	-
Byte $03	länge	--	nummer		--	--	--	--	Name	num.	--	-
Byte $04	Vol.	--	Record-		Vol.	--	Vol.	Vol.	Vol.	Rec.	Vol.	Vol.
Byte $05	Drv.	--	offset		Drv.	Drv.	Drv.	Drv.	Drv.	off.	Drv.	Drv.
Byte $06	Slot	--	Puffer-		Slot	Slot	Slot	Slot	Slot	--	Slot	Slot
Byte $07	Typ	--	länge		--	--	--	--	--	--	--	-
Byte $08	Name	--	Byte oder		Name	--	Adresse		des	--	--	Name
Byte $09	adr.	--	Pufferadr.		adr.	--	Filenamens			--	--	adr.
Byte $0A		F e h l e r c o d e			n a c h		d e r		A u s f ü h r u n g			
Byte $0B					unbenutzt							
Byte $0C		Lo-Byte der Adresse des File-Manager-Arbeitsbereichs (45 Byte)										
Byte $0D		Hi-Byte der Adresse des File-Manager-Arbeitsbereichs										
Byte $0E		Lo-Byte des TSL-Puffers				--	TSL-Puffer Lo			--	--	TSL-Lo
Byte $0F		Hi-Byte des TSL-Puffers				--	TSL-Puffer Hi			--	--	TSL-Hi
Byte $10	--	Daten-Puffer Lo			--	--	--	--	--	--	--	Daten-
Byte $11	--	Daten-Puffer Hi			--	--	--	--	--	--	--	Puffer

Da der Platz in der Tabelle beschränkt ist, besprechen wir einige Positionen genauer. Für alle Befehle gilt, daß in Position 0 der Befehlscode und an Position C/D ein Zeiger (Lo/Hi) auf den Arbeitsbereich von 45 Bytes für den File-Manager angegeben werden müssen. Position A enthält nach der Ausführung den Fehlercode. Bei einem Fehler ist zusätzlich das Carry-Bit gesetzt.
Mögliche Fehler sind:
$00 – kein Fehler (keine eigentliche Meldung)
$01 – Sprache nicht vorhanden (LANGUAGE NOT AVAILABLE)
$02 – ungültiger Befehlscode
$03 – ungültige Untergruppe bei READ und WRITE
$04 – Diskette schreibgeschützt (WRITE PROTECTED)
$05 – Fileende erreicht (EOF)
$06 – File nicht gefunden (FILE NOT FOUND)
$07 – falsche Volume-Nummer (VOLUME MISMATCH)
$08 – I/O Fehler allgemein (I/O ERROR)
$09 – Diskette voll (DISK FULL)
$0A – File gesperrt (FILE LOCKED)

OPEN öffnet eine Datei. Slot, Laufwerk und Volume-Nummer ($00 = alle) müssen übergeben werden. Position 7 enthält den Typ des Files:
$00 – Textfile
$01 – Integer-BASIC Programm
$02 – Applesoft-BASIC Programm
$04 – Binärfile
$08 – Typ S
$10 – Typ R(elokatibel)
$20 – Typ A
$40 – Typ B
Ist Bit 7 zusätzlich gesetzt, ist der File „gelockt".

Position 8 und 9 enthalten einen Zeiger (Lo/Hi) auf den Filenamen, die Positionen E und F einen Zeiger (Lo/Hi) zum TSL-Puffer. Wollen Sie einen Random-Access-Textfile öffnen, muß in den Positionen 2 und 3 die Recordlänge eingetragen werden, ansonsten ein $0000.
Open richtet einen File-Puffer ein und liest den (ersten) TSL-Sektor. In Position 7 wird der tatsächlich gefundene Filetyp abgelegt. Wurde der File nicht gefunden, hängt die Reaktion vom X-Register ab. War das X-Register beim Aufruf des File-Managers Null ($00), wird ein File angelegt, ansonsten erfolgt die Fehlermeldung „FILE NOT FOUND".

CLOSE schließt einen File, indem der letzte Datensektor aus dem Puffer auf die Diskette geschrieben wird sowie VTOC (VOLUME TABLE OF CONTENTS = Diskettenbelegung) und TSL aktualisiert werden. Der File-Puffer wird allerdings nicht freigegeben.

READ und **WRITE** unterscheiden sich nur im Befehlscode in Position 0. Sie lesen oder schreiben ein einzelnes Byte oder eine Gruppe von Bytes von bzw. auf Diskette. Zur näheren Spezifizierung muß eine Untergruppe in Position 1 definiert sein:
$00 – keine Funktion ausführen
$01 – nur 1 Byte lesen/schreiben. Es wird die augenblickliche Position im File benutzt, das Datenbyte wird nach bzw. aus Position 8 transportiert.
$02 – mehrere Bytes lesen/schreiben ab der augenblicklichen Fileposition. In Position 8 und 9 muß die Anfangsadresse der Daten stehen, in Position 6 und 7 die Zahl der Bytes. Beim Schreiben (WRITE) muß diese Zahl um Eins (1) zu klein angegeben werden.
$03 – erst positionieren, dann wie $01. An den Parameter-Positionen 2/3 muß die Recordnummer, an 4/5 der Recordoffset angegeben werden (siehe auch POSITION).
$04 – erst positionieren, dann wie $02. An den Parameter-Positionen 2/3 muß die Recordnummer, an 4/5 der Recordoffset angegeben werden (siehe auch POSITION).
Bei allen Untergruppen wird der Zeiger auf die Fileposition hinter das zuletzt gelesene/geschriebene Byte gesetzt und an den Positionen 2–5 angezeigt. Vor READ/WRITE ist der File mit OPEN zu öffnen.

DELETE markiert im Catalog der Diskette einen File als gelöscht. DELETE führt selbst ein OPEN durch. Bis auf den Befehlscode sind alle Parameter wie bei OPEN. Setzen sie das X-Register vor dem Aufruf auf $00.

CATALOG gibt den Catalog der Diskette auf der aktiven Ausgabeeinheit (zumeist Bildschirm) aus. Außer Slot und Laufwerk ist nur wie immer die Adresse des Arbeitsbereichs an Position C/D anzugeben.

LOCK und **UNLOCK** setzen bzw. löschen das Bit 7 des Filetyps im Catalog der Diskette. Da auch hier ein automatisches OPEN erfolgt, sollte das X-Register gleich Null sein. Alle Parameter entsprechen dem OPEN-Befehl.

RENAME benennt einen File um. Während in der Parameterliste die Positionen 8/9 auf den alten Namen zeigen, muß in Position 2/3 ein Zeiger (Lo/Hi) auf

den neuen Namen stehen. Da auch hier ein OPEN durchgeführt wird, sind die übrigen Parameter wieder identisch wie bei OPEN.

POSITION stellt den Datenzeiger auf eine beliebige Position im mit OPEN geöffneten File. Vor dem ersten READ oder WRITE ist immer ein POSITION erforderlich, wenn nicht die Untergruppen $03 oder $04 benutzt werden. Die Position wird bestimmt, indem die gewünschte Recordnummer (Position 2/3) mit der Recordlänge (aus dem OPEN) multipliziert wird. Für eine Position innerhalb eines Records kann zusätzlich ein Offset (Position 4/5) zum Recordbeginn angegeben werden. Der Datenzeiger wird auf das erste Byte im File gesetzt, indem an den Positionen 2–5 jeweils eine $00 eingetragen wird.

INIT formatiert eine ganze Diskette, legt einen leeren Catalog und eine Belegungsliste (VTOC) an und schreibt ein Abbild des DOS auf die Spuren 0–2. An den Positionen 4–6 sind Volume-Nummer, Slot und Laufwerk anzugeben, an Position 2 zusätzlich das Hi-Byte der ersten DOS-Speicherseite (normal $9D).

VERIFY liest einen File komplett über die RWTS, ohne ihn permanent abzuspeichern. Auf diese Weise wird seine Lesbarkeit überprüft. Die Parameter entsprechen dem OPEN, da dieses automatisch miterfolgt.

10. Mehr PRO als CONTRA

Mit der Einführung von ProDOS hat die Firma Apple für den Assembler-Programmierer eine völlig neue Betriebssystem-Umgebung geschaffen, die ein radikales Umdenken im Vergleich zum alten DOS 3.3 mit sich bringt. Beim DOS 3.3 gab es – mit der Ausnahme einiger Vektoren auf der Seite 3 – keine definierte Schnittstelle zwischen Assembler-Programmen und dem DOS. Da DOS 3.3 aber über viele Jahre konstant blieb, konnte der Programmierer direkt in verschiedene Routinen wie z.B die RWTS oder den File-Manager hineinspringen und sicher sein, daß alles funktionierte. ProDOS ist dagegen nicht adressenkonstant. Bis heute sind wenigstens 4 offizielle Versionen (1.0, 1.0.1, 1.0.2, 1.1.1) auf dem Markt, die sich alle durch mehr oder minder große Verschiebungen auszeichnen. Die Entwickler von ProDOS haben aber für Abhilfe gesorgt, indem sie eine genau definierte und konstante Schnittstelle in das ProDOS einbauten, das Machine Language Interface (MLI, Maschinensprache-Schnittstelle).

10.1 Gute Kontakte per Schnittstelle

Alle direkten Aufrufe des ProDOS sollten über die genormte MLI-Schnittstelle erfolgen. Die Firma Apple hat im „Technical Reference Manual" einen MLI-Aufruf wie folgt definiert:

```
     JSR  $BF00      ; alleinige Einsprungadresse MLI
     HEX  COMMAND    ; Funktionscode
     ADR  PARMBLOCK  ; Adresse des Parameterblocks
     BEQ  KEINFEHL   ; Carry-Flag =0 / Z-Flag = 1, wenn kein Fehler
FEHL ...             ; Fehlerbehandlung, Fehlercode im Akkumulator

PARMBLOCK HEX PARMCOUNT ; Parameter-Anzahl
          HEX  PARM1    ; 1. Parameter (1 Byte)
```

```
ADR  PARM2      ; 2. Parameter (2 Bytes)
HEX  ....       ; N. Parameter
```

Der PARMBLOCK kann irgendwo im Speicher stehen. Das erste Byte steht für die Anzahl der Parameter, die nicht mit der Anzahl der benutzten Bytes identisch sein muß, denn ein Parameter kann auch zwei oder drei Bytes umfassen.

Der Aufruf des MLI muß stets mit einem JSR MLI erfolgen. ProDOS schiebt intern die Rücksprungadresse weiter, sodaß der Rücksprung immer hinter die Adresse des Parameterblocks erfolgt. Für COMMAND ist ein Byte für den aufgerufenen Befehl einzusetzen, das Sie der folgenden Aufstellung entnehmen können. Die beiden folgenden Bytes zeigen im Lo-Hi-Format auf den jeweiligen PARMBLOCK für diesen Aufruf.
MLI-intern wird zuerst das COMMAND überprüft. Bei einem nicht definierten COMMAND folgt ERR 01: BAD SYSTEM CALL NUMBER. Danach wird die Anzahl der Parameter (PARMCOUNT) geprüft. Bei Nichtübereinstimmung folgt ERR 04: BAD SYSTEM CALL PARAMETER COUNT. Ganz allgemein wird ein Ausführungsfehler dadurch angezeigt, daß das Carry-Bit gesetzt und die Zero-Flag gelöscht ist. Der Akkumulator enthält die Fehlernummer. Eine Fehlerbehandlungs-Routine müssen Sie selber schreiben. X- und Y-Register bleiben bei jedem MLI-Aufruf erhalten, da sie zwischengespeichert werden.

Die folgende Aufstellung enthält alle zur Zeit gültigen MLI-Aufrufe und gibt die Zahl und den Aufbau des Parameterblocks an. Als Namen wurden die von Apple eingeführten Bezeichnungen verwendet.

$40: ALLOCATE INTERRUPT

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
Service-Start - 2 Byte-Adresse
```

ProDOS kann bis zu vier Interrupt-Routinen verwalten, die von Ihnen geschrieben und über diesen Aufruf dem ProDOS angemeldet werden müssen. Der Service-Start wird auf den nächsten freien Platz der INTERRUPT TABLE eingetragen, die Referenz-Nummer im Bereich 1-4 wird zurückgeliefert. Fehler: ERR 25: INTERRUPT TABLE FULL (wenn bereits alle vier möglichen Startadressen belegt wurden) sowie ERR 53: INVALID SYSTEM CALL PARAMETER (Startadresse kleiner als $0100).

$41: DEALLOCATE INTERRUPT

```
PARMCOUNT = 1
Referenz-Nummer - 1 Byte
```

In der INTERRUPT TABLE wird die Startadresse der bezeichneten Interrupt-Routine gelöscht.
Fehler: ERR 43: INVALID REFERENCE NUMBER (Referenz-Nummer nicht im Bereich 1-4).

$65: REBOOT (= QUIT)

```
PARMCOUNT = 4
.....
```

Die Routine REBOOT wird aus der Language Card kopiert und ausgeführt. Dieses COMMAND ist nachträglich eingebaut worden. Die vier Parameter für dieses COMMAND sind „unecht". Sie sollten 6 mal $00 benutzen (2 Zweier- und 2 Einergruppen).
REBOOT erkärt den ganzen Speicher als unbelegt und fragt den Benutzer nach dem Namen eines System-Programms. Dieses wird anschließend geladen und gestartet.

$80: READ BLOCK

```
PARMCOUNT = 3
Unitnummer - 1 Byte (DSSS 0000)
Zielpuffer - 2 Byte-Adresse
Blocknummer - 2 Bytes
```

Liest einen Block (= 2 Sektoren) von einem externen Datenspeicher ab „Zielpuffer" in den Speicher.
Vor dem Aufruf wird überprüft, ob der Zielbereich als frei deklariert ist. Bei Zieladressen, die größer als $BFFF sind, ergibt dieser Test Unsinn.
Bei der Unitnummer ist in Bit 7 die Laufwerksnummer D (0 = Laufwerk 1, 1 = Laufwerk 2) und in den Bits 4–6 die Slotnummer S (1–7) anzugeben. S6,D1 ⇒ %0110 0000 = $60
Fehler ERR 56: BAD BUFFER ADDRESS (Zielgebiet belegt), ERR 28: NO DEVICE CONNECTED (Unitnummer eines nicht existierenden Datenspeichers) sowie ERR 27: I/O ERROR.

$81: WRITE BLOCK

```
PARMCOUNT = 3
Unitnummer - 1 Byte (DSSS 0000)
Quellpuffer - 2 Byte-Adresse
Blocknummer - 2 Bytes
```

Der entsprechende Block mit dem Speicherinhalt ab „Quellpuffer" wird auf den angesprochenen Datenspeicher geschrieben.
Bei der Unitnummer ist in Bit 7 die Laufwerksnummer D (0 = Laufwerk 1, 1 = Laufwerk 2) und in den Bits 4–6 die Slotnummer S (1–7) anzugeben. S6,D2 ⇒ %1110 0000 = $E0
Fehler wie bei READ BLOCK, nur kommt noch ERR 2B: DISK WRITE PROTECTED hinzu.

$82: GET TIME

```
PARMCOUNT = 0
```

Dieses COMMAND wurde nachträglich „eingeflickt" und bewirkt einen Aufruf von $BF06 – hier sollte dann ein Sprung zur Uhrenroutine (Standard: $F142) stehen.
Die offizielle Uhr hat folgende Ausgabe:
$BF90/91: Lo/Hi von JJJJJJJM MMMTTTTT. J = Jahr nach 1900, M = Monat (1-12), T = Tag (1-31).
$BF92/93: Lo/Hi von SSSSSSSS MMMMMMMM. S = Stunde, M = Minute
Fehler: keine.

$C0: CREATE

```
PARMCOUNT = 7
Path-Name - 2 Byte-Adresse
Access - 1 Byte
File-Type - 1 Byte
Aux-Type - 2 Bytes
Storage-Type - 1 Byte
Creation Date - 2 Bytes
Creation Time - 2 Bytes
```

Mit diesem COMMAND werden Daten- oder Directory-Files erstellt. Soll eine bereits existierende Datei neu angelegt werden, muß sie erst mit $C1 DESTROY gelöscht werden.
Access enthält eine Bitmaske mit folgender Bedeutung:
Bit 7 gesetzt, wenn Datei gelöscht werden kann

Bit 6 gesetzt, wenn Datei umbenannt werden kann
Bit 5 gesetzt, wenn Datei Back-Up benötigt
Bit 4-2 ungenutzt
Bit 1 gesetzt, wenn auf Datei geschrieben werden kann
Bit 0 gesetzt, wenn von Datei gelesen werden kann.
CREATE setzt Bit 5 automatisch.

File-Type umfaßt folgende Haupt-Filetypen:

Typ	Name	Bedeutung
$00		ohne Typ
$01	BAD	Datei mit Fehlerblock
$04	TXT	ASCII-Textdatei mit Bit 7 = 0
$06	BIN	Binärfile
$0F	DIR	Subdirectory
$19	ADB	Appleworks Datenbank
$1A	AWP	Appleworks Textverarbeitung
$1B	ASP	Appleworks Tabellenkalkulation
$EF	PAS	ProDOS PASCAL
$F0	CMD	Befehls-Datei
$F1	bis $F8	Benutzerdefiniert
$FA	INT	Integer BASIC File (?)
$FB	IVR	Integer BASIC Variablen (?)
$FC	BAS	Applesoft BASIC File
$FD	VAR	Applesoft BASIC Variablen
$FE	REL	relokatives Assemblermodul von EDASM
$FF	SYS	System-File

Aux-Type enthält je nach File-Type unterschiedliche Angaben:
für TXT die Record-Länge (Lo/Hi),
für BIN, BAS, VAR und SYS die Lade-Adresse (Lo/Hi).
Ansonsten wird diese Information nicht benötigt.

Der Parameter „Path-Name“ enthält die Adresse, auf der der Pathname selber in ASCII mit vorangestelltem Längenbyte stehen muß. Der letzte Name im angegebenen Path ist der Name des zukünftigen Files.

Die Unterscheidung zwischen „Datenfile“ und „Subdirectory“ hängt nur vom angegebenen Storage Type ab: für $0D wird ein Subdirectory Key Block erstellt, für 0/1/2/3 wird ein Datenfile erstellt, der immer den Storage Type 1 (Sämling) hat.
Wenn DATE und TIME angegeben sind (<> 00), werden die Werte aus dem Parameterblock benutzt, ansonsten wird die System-Zeit eingesetzt.
Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR 40: INVALID PATHNAME, ERR 44/45: DIRECTORY/VOLUME NOT FOUND, ERR 47: DUPLICATE FILENAME, ERR 48: VOLUME FULL, ERR 49: VOLUME DIRECTORY FULL, ERR 4B: UNSUPPORTED STORAGE TYPE (für CREATE mit Storage Types <> 0/1/2/3/$D), ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$C1: DESTROY

```
PARMCOUNT = 1
Path-Name - 2 Byte-Adresse
```

Der File, auf dessen Namen „Path-Name“ zeigt (siehe CREATE), wird aus dem Directory entfernt, die zum File gehörigen Blocks werden wieder freigegeben.
DESTROY ist nicht möglich, wenn
a) der File OPEN ist,
b) im Access-Byte des File-Eintrags Bit 7 nicht gesetzt ist,
c) der File ein Subdirectory ist, das seinerseits noch weitere File-Einträge enthält.
In diesen Fällen folgt ERR 4E: FILE ACCESS ERROR, ERR 50: FILE IS OPEN
Weitere Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR: 40 INVALID PATHNAME, ERR: 44 DIRCTORY NOT FOUND, ERR: 45 VOLUME NOT FOUND, ERR: 46 FILE NOT FOUND, ERR: 4A INCOMPATIBLE FILE FORMAT

$C2: RENAME

```
PARMCOUNT = 2
Path-Name - 2 Byte-Adresse
Neuer Path-Name - 2 Byte-Adresse
```

RENAME ist für Volumes, Subdirectories und Datenfiles möglich. Zur Definition der Path-Namen siehe unter CREATE.

Der Unterschied zwischen neuem und altem Path muß im letzten Namensteil beider Paths auftreten, ansonsten folgt ERR 40: INVALID PATHNAME. Alle anderen Namensteile der Paths müssen gleich sein.
Für Volumes ist RENAME nur möglich, wenn auf dem Volume kein File (auch das Directory selber) OPEN ist, ansonsten folgt ERR 50: FILE IS OPEN. Für Subdirectories/Datenfiles wird zusätzlich geprüft, ob der Access des Files das entsprechende Bit (Bit 6) gesetzt hat. Ist das nicht der Fall, folgt ERR 4E: FILE ACCESS ERROR.
Existiert bereits ein File mit dem neuen Namen, folgt ERR 47: DUPLICATE FILENAME.
Weitere Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR: 40 INVALID PATHNAME, ERR: 44 DIRCTORY NOT FOUND, ERR: 45 VOLUME NOT FOUND, ERR: 46 FILE NOT FOUND, ERR: 4A INCOMPATIBLE FILE FORMAT, ERR: 4B UNSUPPORTED STORAGE TYPE, ERR: 57 DUPLICATE VOLUME

$C3: SET FILE INFO

```
PARMCOUNT = 7
Path-Name - 2 Byte-Adresse
Access - 1 Byte
File-Type - 1 Byte
Aux-Type - 2 Bytes
Null-Feld - 3 Bytes
Modification Date - 2 Bytes
Modification Time - 2 Bytes
```

Dieses COMMAND verändert die Attribute (Eigenschaften) eines Files. SET FILE INFO kann für Volume Directories nicht angewandt werden; ERR 40: INVALID PATHNAME. Mit diesem COMMAND kann z.B. der Access eines Files neu gesetzt werden, da das Sperren eines Files gegen ein SET FILE INFO nicht möglich ist.
Zu Path-Name, Access, File-Type, Aux-Type siehe unter CREATE. Das Null-Feld kann beliebige Werte annehmen. Es existiert lediglich, um für die ersten 7 Parameter dasselbe Format des PARMBLOCK wie für GET FILE INFO zu erhalten.
Wenn DATE und TIME spezifiziert sind, werden diese Angaben benutzt, ansonsten wird die System-Zeit eingesetzt.
Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR 40: INVALID PATHNAME, ERR 44/45/46: DIRECTORY/VOLUME/FILE NOT FOUND, ERR: 4A INCOMPATIBLE FILE FORMAT, ERR: 4B

UNSUPPORTED STORAGE TYPE, ERR: 4E FILE ACCESS ERROR, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$C4: GET FILE INFO

```
PARMCOUNT = $0A (10)
Path-Name - 2 Byte-Adresse
Access - 1 Byte-Resultat
File-Type - 1 Byte-Resultat
Aux-Type - 2 Byte-Resultat / TOTAL BLOCKS
Storage-Type - 1 Byte-Resultat
BLOCKS USED - 2 Byte-Resultat / TOTAL BLOCKS USED
Modification Date - 2 Byte-Resultat
Modification Time - 2 Byte-Resultat
Create Date - 2-Byte Resultat
Create Time - 2-Byte Resultat
```

Dieses COMMAND stellt das Gegenstück zu SET FILE INFO dar. Es liest die Attribute einer Datei in die Parameterliste ein. Von Ihnen muß lediglich PARMCOUNT und der Zeiger auf den Path-Namen gesetzt sowie dahinter der benötigte Platz freigehalten werden.
Für Subdirectories/Datenfiles enthält der Aux-Type den Aux-Type und BLOCKS USED gibt die Anzahl der vom File belegten Blocks an.
Dem Null-Feld von SET FILE INFO entsprechen hier Storage-Type und BLOCKS USED. Diese beiden Parameter sind mit SET FILE INFO nicht veränderbar.
Für Volume Directories wird anstelle des Aux-Type die Gesamtkapazität des Volumes (TOTAL BLOCKS) und für BLOCKS USED die Gesamtzahl der auf dem Volume belegten Blocks (TOTAL BLOCKS USED) zurückgeliefert.
Zu den Definitionen der Parameter siehe CREATE. Für Storage-Type gilt:
$0 = gelöschter File
$1 = Seedling (Sämling) mit max. 1 Datenblock
$2 = Sapling (Schößling) mit einem Indexblock
$3 = Tree (Baum) mit einem Masterblock
$D = Subdirectory
Fehler: ERR: 27 I/O ERROR, ERR 40: INVALID PATHNAME, ERR 44/45/46: DIRECTORY/VOLUME/FILE NOT FOUND, ERR: 4A INCOMPATIBLE FILE FORMAT, ERR: 4B UNSUPPORTED STORAGE TYPE, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$C5: ON LINE

```
PARMCOUNT = 2
Unitnummer - 1 Byte (DSSS 0000) oder $00 („Alle")
Zielpuffer - 2 Byte-Adresse
```

ON LINE schreibt den Volume-Namen der/des adressierten Units (Unitnummer siehe READ BLOCK) ab der Adresse „Zielpuffer" in den Speicher. Wenn die Unitnummer mit $00 angegeben ist, werden sämtliche Units aus der DEVICE TABLE bearbeitet.
Bei einer spezifizierten Unitnummer (<> $00) müssen $10, ansonsten $100 Bytes bereitgestellt werden. Ist der entsprechende Speicherbereich als belegt deklariert, folgt ERR 56: BAD BUFFER ADDRESS.
Pro Unit werden 16 Byte ausgegeben:
0. Byte: DSSSNNNN (D = Drive, S = Slot, N = Länge Volume-Name
1.–15. Byte: Fehlercode oder Volume-Name ohne Schrägstrich
Bei ERR: 57 enthält Byte 3 die Nummer des anderen Units mit gleichem Volume-Namen. Das Ende der Einträge ist durch $00 in Byte 0 und 1 gekennzeichnet.

Fehler: ERR: 28 NO DEVICE CONNECTED, ERR: 2E DISK SWITCHED AND FILE OPEN, ERR: 45 VOLUME NOT FOUND, ERR 52: NOT A PRODOS VOLUME, ERR 57: DUPLICATE VOLUME (wenn zwei Volumes den selben Namen haben und auf mindestens einem der beiden Volumes ein File OPEN ist).

$C6: SET PREFIX

```
PARMCOUNT = 1
Path-Name - 2 Byte-Adresse
```

Dieses COMMAND setzt ein PREFIX und sucht das entsprechende Volume bzw. die entsprechende Subdirectory.
Path-Name siehe CREATE. Ist das erste Zeichen ein „/", so wird das alte Prefix ersetzt, ansonsten wird der neue Path-Name an das alte Prefix zu einem neuen Prefix angehängt.
SET PREFIX ist auch mit einer Länge von $00 möglich, d.h. der Pathname besteht nur aus einem „/". In diesem Fall wird das PREFIX gelöscht.

Fehler: ERR 40: INVALID PATHNAME („letztes File im Path-Name ist kein Subdirectory" oder „Gesamt-Prefix länger als $40 Zeichen"), ERR: 44/45/46 DIRECTORY/VOLUME/FILE NOT FOUND, ERR: 4A INCOMPATIBLE

FILE FORMAT, ERR: 4B UNSUPPORTED STORAGE TYPE, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$C7: GET PREFIX

```
PARMCOUNT = 1
Zielpuffer – 2 Byte-Adresse
```

Das PREFIX wird in den Zielpuffer ($40 Bytes Platzbedarf) kopiert (beginnend mit Längenbyte und ein letzter „/" angehängt). Ist kein PREFIX gesetzt, folgt Rücksprung ohne Fehler.
Fehler: ERR 56: BAD BUFFER ADDRESS.

$C8: OPEN

```
PARMCOUNT = 3
Path-Name – 2 Byte-Adresse
Filepuffer – 2 Byte-Adresse
Referenz-Nummer – 1 Byte Resultat
```

OPEN sucht eine Datei und reserviert für sie einen I/O-Puffer.
Der File muß bereits existieren, ansonsten folgt ERR 46: FILE NOT FOUND. In der System Bit Map wird der Filepuffer mit einem Platzbedarf von $400 Bytes ab der angegebenen Startadresse (Lo-Byte muß $00 sein !!) geprüft und belegt. Ist der entsprechende Platz nicht frei, folgt ERR 56: BAD BUFFER ADDRESS.
Die Referenz-Nummer (1–8) wird in den Parameterblock eingetragen. Über diese Nummer finden alle zukünftigen Aktionen mit diesem File statt. Sie muß deshalb vom Programm zwischengespeichert werden. Es können maximal 8 Files gleichzeitig geöffnet sein.
Fehler: ERR: 27 I/O ERROR, ERR 40: INVALID PATHNAME, ERR: 42 MAXIMUM NUMBER OF FILES OPEN, ERR 44/45/46: DIRECTORY/ VOLUME/FILE NOT FOUND, ERR: 4B UNSUPPORTED STORAGE TYPE, ERR 50: FILE IS (ALREADY) OPEN, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$C9: NEWLINE

```
PARMCOUNT = 3
Referenz-Nummer – 1 Byte
AND-Maske – 1 Byte
Ende-Zeichen – 1 Byte
```

Das Command NEWLINE kann nur auf OPEN Files angewandt werden und setzt eine End-Bedingung für READ. Wenn NEWLINE aktiv ist, wird bei jedem Byte, das via READ gelesen wird, zuerst mit dem Masken-Byte ein AND durchgeführt. Danach erfolgt ein Vergleich mit dem Ende-Zeichen. Bei Übereinstimmung wird READ beendet. Das BASIC.SYSTEM setzt für Textfiles die Maske auf $7F und das Ende-Zeichen auf $0D. Damit endet jedes READ spätestens bei einem <CR> ($8D oder $0D). Ist die Maske auf $00 gesetzt, ist NEWLINE deaktiviert.
Fehler: ERR 43: INVALID REFERENCE NUMBER.

$CA: READ

```
PARMCOUNT = 4
Referenz-Nummer - 1 Byte
Zielpuffer - 2 Byte-Adresse
Anforderungs-Zähler - 2 Bytes
Übertragungs-Zähler - 2 Bytes Resultat
```

READ kann nur auf OPEN Files angewendet werden. Es werden die im Anforderungs-Zähler angegebenen Bytes ab der momentanen „Position im File" in den Zielpuffer gelesen.
Vor READ wird EOF (END OF FILE) geprüft und die zu übertragende Anzahl an Bytes gegebenenfalls herabgesetzt. Ist das EOF bereits erreicht, folgt ERR 4C: END OF FILE ENCOUNTERED.
Bei aktivem NEWLINE endet READ ebenfalls, wenn das Ende-Zeichen erkannt wurde. NEWLINE setzt die Lesegeschwindigkeit bei größeren Datenmengen deutlich herab, da READ für Anforderungen von kompletten Blöcken ansonsten über einen „Schnellgang" verfügt, der direkt ins Zielgebiet und nicht in einen Zwischenpuffer liest.
Wenn nicht der gesamte Zielpuffer als frei markiert ist, folgt ERR 56: BAD BUFFER ADDRESS.
Nach dem Ende von READ wird im Parameterblock der Übertragungs-Zähler mit der Anzahl der tatsächlich transferierten Bytes besetzt. READ verschiebt die „Position im File" um die Anzahl der gelesenen Bytes.
Fehler: ERR: 27 I/O ERROR, ERR 43: INVALID REFERENCE NUMBER, ERR 4E: FILE ACCESS ERROR (Bit 0 nicht gesetzt), ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$CB: WRITE

```
PARMCOUNT = 4
Referenz-Nummer - 1 Byte
Quellpuffer - 2 Byte-Adresse
Anforderungs-Zähler - 2 Bytes
Übertragungs-Zähler - 2 Bytes
```

WRITE kann nur auf OPEN Files angewendet werden. Es werden die im Anforderungs-Zähler angegebenen Bytes ab der momentanen „Position im File" aus dem Quellpuffer in den File geschrieben. Der File wird gegebenenfalls durch Anhängen weiterer Blocks und Erhöhung des Storage Type erweitert. Auch der EOF erhöht sich.
Das MLI versucht immer, alle angeforderten Bytes auf den File zu schreiben. Ein Abbruch kann nur durch eine Fehlerbedingung erfolgen.
Bis zu 512 Bytes werden zunächst in einem Zwischenpuffer gehalten. Erst wenn der Puffer voll ist, die Datei mit CLOSE geschlossen oder der Puffer mit FLUSH entleert wird, werden sie tatsächlich auf die Diskette geschrieben. Damit alle Dateien vollständig sind, sollten Sie jede Datei schließen, bevor Sie eine Diskette wechseln oder den Rechner ausschalten.
Nach Beendigung von WRITE wird in den Übertragungs-Zähler im Parameterblock die Anzahl der tatsächlich geschriebenen Bytes eingetragen. Während WRITE verschiebt sich die „Position im File" um die Anzahl der geschriebenen Bytes.

Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR 43: INVALID REFERENCE NUMBER, ERR 48: VOLUME FULL, ERR 4E: FILE ACCESS ERROR (Bit 1 nicht gesetzt), ERR: 56 BAD BUFFER ADDRESS, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$CC: CLOSE

```
PARMCOUNT = 1
Referenz-Nummer - 1 Byte oder $00 („Alle")
```

Dieses COMMAND schreibt den I/O-Puffer auf den OPEN File und aktualisiert den Dateieintrag. Der File-Puffer wird wieder freigegeben. Bei einer spezifizierten Referenz-Nummer muß das File OPEN sein, sonst folgt ERR 43: INVALID REFERENCE NUMBER. Bei einem „Alle" (= $00) werden alle OPEN Files geschlossen, wenn gleichzeitig das LEVEL-Byte ($BF94) auf $00 gesetzt wurde. CLOSE ruft FLUSH auf.

Bei einem „Alle" werden Fehlernummern zwischengespeichert, d.h es werden alle OPEN Files wirklich bearbeitet.
Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$CD: FLUSH

```
PARMCOUNT = 1
Referenz-Nummer - 1 Byte oder $00 („Alle")
```

Dieses COMMAND arbeitet wie CLOSE, nur daß der File nicht geschlossen und sein Puffer nicht freigegeben wird. Wenn längere Zeit nicht auf ein File geschrieben wird, können mit FLUSH zwangsweise der I/O-Puffer geleert und die Dateieinträge aktualisiert werden. Bei einem „Alle" muß $BF94 (LEVEL) $00 sein, um alle Files zu bearbeiten.
Fehler: ERR: 27 I/O ERROR, ERR: 2B WRITE PROTECTED, ERR: 43 INVALID REFERENCE NUMBER, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$CE: SET MARK

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
Position gewünscht - 3 Bytes (Lo/Mid/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es setzt die „Position im File", die für den nächsten Zugriff mit READ oder WRITE benutzt wird. Wird vom Benutzer eine Position spezifiziert, die hinter dem momentanen EOF des Files liegt, folgt ERR 4D: POSITION OUT OF RANGE.
Das erste Byte der Datei hat die Position $000000. Wegen der möglichen Maximallänge eines Files muß die „Position im File" eine 3-Byte-Zahl sein, wobei das niederwertigste Byte vorne steht.
Fehler: ERR: 43 INVALID REFERENCE NUMBER, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$CF: GET MARK

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
Position im File - 3 Byte-Resultat (Lo/Mid/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es kopiert die momentane „Position im File", also den Start des nächsten READ oder WRITE, in den PARMBLOCK.
Fehler: ERR 43: INVALID REFERENCE NUMBER

$D0: SET EOF

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
neuer EOF - 3 Bytes (Lo/Mid/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es setzt den EOF (End Of File ≙ Dateigröße) eines Files.
SET EOF ist nur für Datenfiles und nicht für Subdirectories möglich. Ist WRITE nicht erlaubt (Bit 1 im Access-Byte gelöscht), folgt ERR 4E: FILE ACCESS ERROR.
Bei einer Vergrößerung des EOF entstehen sogenannte „Loch-Dateien", deren Blöcke beim nächsten WRITE belegt werden.
Bei einer Verkleinerung des Files werden Blöcke aus dem File wieder freigegeben und innerhalb der Master/Indexblock(s) gelöscht.
Fehler: ERR: 27 I/O ERROR, ERR: 43 INVALID REFERENCE NUMBER, ERR: 4D POSITION OUT OF RANGE, ERR: 5A FILE STRUCTURE DAMAGED (BIT-MAP defekt).

$D1: GET EOF

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
EOF - 3 Byte-Resultat (Lo/Mid/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es kopiert den momentanen EOF des Files in den PARMBLOCK.
Fehler: ERR 43: INVALID REFERENCE NUMBER

$D2: SET BUF

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
Pufferstart - 2 Byte-Adresse (Lo/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es setzt einen neuen Filepuffer, wobei der Inhalt des alten in den neuen Filepuffer

kopiert und der alte Filepuffer wieder freigegeben wird.
Alter und neuer Puffer dürfen sich nicht überschneiden, das Lo-Byte der Adresse muß $00 sein. Ist der Bereich des neuen Filepuffers ($400 Bytes) nicht vollständig frei, folgt ERR 56: BAD BUFFER ADDRESS.
Fehler: ERR: 43 INVALID REFERENCE NUMBER

$D3: GET BUF

```
PARMCOUNT = 2
Referenz-Nummer - 1 Byte
Pufferstart - 2 Byte-Adresse Resultat (Lo/Hi)
```

Dieses COMMAND kann nur auf OPEN Files angewandt werden. Es schreibt die Startadresse des zugehörigen Filepuffers in den PARMBLOCK.
Fehler: ERR 43: INVALID REFERENCE NUMBER.

10.2 Wie spreche ich mit ProDOS?

Unter DOS 3.3 ist es eine gängige Praxis, Diskettenbefehle mit einem vorangestellten Ctrl-D als Zeichenkette über COUT auszugeben. Unter ProDOS funktioniert dieses Verfahren **nicht** mehr. Wenn nur PRODOS.SYSTEM im RAM ist, gibt es keine Möglichkeit mehr, mit Strings einen Diskettenbefehl auszuführen. Befindet sich allerdings auch das BASIC.SYSTEM im Speicher – was zur Ausführung eines Applesoftprogramms notwendig ist –, dann existiert eine Alternative:
a) Sie legen die Zeichenkette (z.B. CATALOG) **ohne** Ctrl-D ab $0200 in den Tastatureingabe-Puffer. Das Bit 7 von jedem Zeichen muß gesetzt sein. Die Zeichenkette muß mit $8D (RETURN) abgeschlossen werden.
b) Sie rufen den Command-Handler auf (JSR $BE03).
Wurde der Befehl korrekt ausgeführt, kehrt die Routine mit gelöschtem Carry-Bit zurück. Bei einem Fehler erfolgt eine Fehlermeldung und ein Programmabbruch.
Die Befehle –, RUN, LOAD, CHAIN, OPEN, READ, WRITE, APPEND, POSITION, EXEC, IN# und PR# sind auf diese Weise **nicht** ausführbar.

Noch von einer weiteren lieben Gewohnheit müssen wir uns unter ProDOS verabschieden. Unter DOS 3.3 konnten Sie die Ein- und Ausgabe des Apple

umlenken, indem Sie die neuen Adressen in die Vektoren CSW ($0036/37) und KSW ($0038/39) schrieben und anschließend die Routine CONNECT (JSR $03EA) aufriefen. Da die Adresse $03EA nicht mehr in Funktion ist, können Sie dieses Verfahren unter ProDOS nicht mehr anwenden.
Wenn das BASIC.SYSTEM im Speicher steht, gibt es folgende Alternative:
a) Sie lassen CSW und KSW unverändert.
b) In den Ausgabe-Vektor $BE30/31 tragen Sie die gewünschte Ausgabeadresse ein (normal COUT1 $FDF0).
c) In den Eingabe-Vektor $BE32/33 tragen Sie die gewünschte Eingabeadresse ein (normal KEYIN $FD1B).
Das ist alles!

10.3 Wo find' ich ein Zuhause?

Wenn Sie ein BASIC-Programm und ein Assembler-Programm gleichzeitig im Speicher halten wollen, benötigen Sie einen sicheren Ort für den Maschinencode. Vor allen Dingen muß er vor dem Überschreiben durch BASIC-Variablen geschützt werden. Wenn der Platz von $0300 bis $03CF nicht ausreicht, müssen wir einen anderen Platz wählen. Unter DOS 3.3 wurde zumeist der RAM-Bereich unter den DOS-Puffern (unterhalb $9600 bei MAXFILES 3) gewählt und dann HIMEM: unter das Programm gesetzt. Auch dieses Verfahren funktioniert unter ProDOS nicht mehr, denn ProDOS hat eine dynamische Pufferverwaltung. Das heißt, daß ProDOS selbsttätig Puffer einrichtet und löscht. Den Befehl MAXFILES gibt es nicht mehr.
Das BASIC.SYSTEM liegt von $9A00 an aufwärts im RAM. Darunter befindet sich von $9600 (HIMEM:) bis $99FF ein Allzweckpuffer. Wird ein File geöffnet (OPEN), wird dieser Allzweckpuffer nach $9200 (neues HIMEM:) verschoben und ab $9600 ein Filepuffer eingerichtet. Jedes weitere OPEN verschiebt den Allzweckpuffer und HIMEM: um weitere 1024 Bytes nach unten (bei 2 offenen Files nach $8E00) und setzt den neuen Filepuffer an die alte Adresse des Allzweckpuffers (= unter den letzten Filepuffer). Lediglich bei einem EXEC-File werden alle Puffer nach unten verschoben und der EXEC-Filepuffer bei $9600 eingerichtet. Wenn Files geschlossen (CLOSE) werden, wird der zugehörige Puffer aufgehoben und alle übrigen Puffer werden wieder nach oben geschoben.

Der einzige sichere Ort für ein Assembler-Programm liegt zwischen dem BASIC.SYSTEM (BI) und den Filepuffern. Das BASIC.SYSTEM kann eine beliebig große Anzahl von Speicherseiten (je 256 Bytes) zwischen sich und dem ersten Puffer freihalten. Die Anzahl der gewünschten Seiten ist in den Akkumulator zu laden und dann GETBUFR ($BEF5) aufzurufen. Wenn das Carry-Bit gelöscht ist, war die Einrichtung des geschützten Bereichs erfolgreich. Der Akkumulator enthält dann das Hi-Byte der unteren Pufferadresse (das Lo-Byte ist immer $00, X- und Y-Register werden zerstört!).
Der erste geschützte Bereich wird immer direkt unterhalb des BASIC.SYSTEM ($9A00) eingerichtet, egal wieviele Filepuffer gerade geöffnet sind. Seine Adresse ist deshalb vorhersehbar und Sie benötigen *keinen* relokativen Code für Ihr Assembler-Programm.
Fordern Sie über GETBUFR weiteren Speicher an, so wird dieser unterhalb des zuletzt reservierten Bereichs eingerichtet.

```
LDA #2          ;512 Bytes freihalten
JSR GETBUFR     ;Anforderung
BCS FEHLER      ;Fehlerbehandlung
STA SPEICHER+1 ;Hi-Byte des Pufferanfangs
LDA #$00
STA SPEICHER    ;Lo-Byte
```

Ab (SPEICHER) können Sie dann Ihr Maschinenprogramm laden. Sie können nachträglich auch noch weitere Programme in den reservierten Bereich laden, da ProDOS ihn intern (in der System Bit Map, auf die wir hier nicht eingehen) als nicht belegt markiert hat.
Sie können den reservierten Bereich nur als Ganzes wieder freigeben. Dazu ist ein JSR FREEBUFR ($BEF8) notwendig. Die Filepuffer werden dann wieder unter das BASIC.SYSTEM gesetzt.

Wenn Sie Maschinenprogramme benutzen wollen, die unter DOS 3.3 geschrieben wurden (praktisch alle Programme dieses Buches), ist zu beachten:
a) Wenn das Programm völlig selbständig arbeitet und keine Aufrufe an das DOS macht (ROM-Aufrufe funktionieren weiterhin), kann es den gesamten Speicher bis $BEFF belegen, da das BASIC.SYSTEM dann nicht benötigt wird.
b) Wenn das Programm DOS-Aufrufe macht, sind die betreffenden Teile völlig umzuschreiben. Entweder muß das MLI benutzt werden – dann ist nur PRODOS.SYSTEM notwendig – oder auch das BASIC.SYSTEM (dann ist der Speicher nur bis $9600 frei).

c) Wenn gleichzeitig auch Applesoft benutzt wird, muß das Maschinenprogramm zwischen dem BASIC.SYSTEM und den Filepuffern „versteckt" werden (siehe oben). Die entsprechenden Routinen sind zu ergänzen, der ORG muß angepaßt werden. Dafür sind alle Teile, die HIMEM: versetzen, zu streichen.

Noch eine Schlußbemerkung: In diesem Buch ist nicht ausreichend Platz, um Beispielprogramme für alle neuen Möglichkeiten des ProDOS zu zeigen. „Apple ProDOS für Aufsteiger, Band 1 und 2" von Ulrich Stiehl, Hüthig Verlag Heidelberg, gibt Ihnen viele kleine und große Anregungen für eigene Programme.

11. Schnell wie der Wind

Dieses ist das letzte Kapitel in dem zweibändigen Kursus „Apple-Assembler lernen". Sie haben bis jetzt fast alles gelernt, was man über die Assemblerprogrammierung des 6502/65C02 wissen muß, und Sie haben sehr viel über Ihren Apple erfahren. Kein Kursus kann alle Themenbereiche einer Sache vollständig abdecken. Zwei Dinge haben wir völlig ausgelassen: die Steuerung des Rechners mit Interrupts und die Anwendung und Programmierung der Maus. Ein dritter Bereich ist noch nicht ganz behandelt worden: die Programmierung von Diskettenzugriffen. Wir haben zwar gelernt, mit DOS 3.3 und ProDOS umzugehen, aber das ist nicht der einzige Weg. Die Betriebssysteme sind ja auch nur wieder Assembler-Programme, derer wir uns bedient haben. Es gibt daneben noch den ganz direkten Weg durch das Ansprechen des Disk-Controllers und seiner Hardware-Adressen.
In diesem Kapitel werden wir nun ein schnelles Kopierprogramm für DOS-, ProDOS-, CP/M- und PASCAL-Disketten schreiben, das in einem Arbeitsgang eine normale 35 Track Diskette formatiert und kopiert. Diese Aufgabe ist nur zu meistern, wenn wir direkt die Hardware des Disketten-Controllers ansteuern. Da Sie nun schon ein recht erfahrener Programmierer sind, werde ich Ihnen keine Programmbeschreibung mehr liefern. Das Listing ist aber reichlich kommentiert, sodaß Sie viele Hilfen zum Verständnis haben.
Ein paar Dinge haben wir noch nicht angesprochen, die unbedingt zum Verstehen eines Kopierprogramms notwendig sind: der Aufbau einer Diskette und die Hardware des Controllers. Vor dem eigentlichen Programm werden wir uns deshalb noch mit diesen Themen beschäftigen müssen.

11.1 Rund und dunkelbraun

Eine Diskette ist eine magnetisierbare Scheibe ähnlich einem Tonband. Die Diskette wird beschrieben und von ihr wird gelesen, während sie sich mit 300 Umdrehungen pro Minute unter einem Magnetkopf herbewegt. Auf diese Weise werden kreisförmige Spuren bearbeitet. Der Kopf kann radial bewegt werden und so verschiedene Spuren erreichen. Beim DISK II Laufwerk von Apple sind es 35 Spuren (= **Tracks**). Der Kopf wird durch einen Schrittmotor gesteuert, der unter der Kontrolle des 6502-Prozessors steht. Der Apple hat also keinen eigenen Disk-Prozessor. Es ist ferner kein Sensor vorhanden, der dem Rechner die absolute Kopfposition mitteilt. DOS und ProDOS ziehen deshalb den Kopf bis an einen mechanischen Anschlag (Spur 0), was das berühmte „Klappern" bewirkt. Von da an werden alle Kopfbewegungen im RAM protokolliert, damit die Betriebssysteme die Kopflage kennen. Auf jede Spur ist außerdem ein Bitmuster geschrieben, in das u.a. auch die Spurnummer codiert ist.

Vor der ersten Benutzung muß eine Diskette formatiert werden. Dabei wird ein festgelegtes Bitmuster auf sie geschrieben, das eine Spur in 16 Datenfelder für je 256 Bytes aufteilt, die als Kreissegmente gleichmäßig auf der Spur verteilt liegen. Eine solche Diskette wird „softsektoriert" genannt im Gegensatz zu den heute nicht mehr gebräuchlichen hardsektorierten Disketten, bei denen jedes Datenfeld mit einem Loch in der Diskette gekennzeichnet war.
Vor jedem Datenfeld liegt noch ein kleines Adressfeld, das die Nummer des Sektors, des Tracks und des Volumes der Diskette enthält. Adress- und Datenfeld bilden zusammen einen **Sektor**.
Vor und hinter dem Adressfeld sowie vor und hinter dem Datenfeld befinden sich noch ein paar Bytes (je 3), die einen Prolog (Header) bzw. Epilog (Trailer) bilden und zur Identifizierung der Felder dienen. Die Sektoren sind eingebettet in eine besondere Art von Bytes, den Synchronisations-Bytes (Syncs), die dafür verantwortlich sind, die Elektronik des Controllers und den Datenstrom auf der sich drehenden Diskette im Gleichtakt zu halten. Es würde an dieser Stelle etwas weit führen, alle Details zu behandeln. Eine gute Beschreibung finden Sie in „Beneath Apple DOS", Quality Software.
Der prinzipielle Aufbau eines Sektors lautet:

```
   5-40 Syncs       Prolog   Volume   Track  Sektor  Prüfsumme Epilog
FF FF FF...FF FF  D5 AA 96  VOL VOL TRK TRK SEC SEC SUM SUM  DE AA EB

   5-8 Syncs        Prolog        D a t e n       Prüfsumme  Epilog
FF FF FF...FF FF  D5 AA AD    342 Datenbytes     SUM SUM    DE AA EB
```

Sie werden sich sicherlich wundern, warum hier 342 Datenbytes angegeben sind, obwohl doch nur 256 Bytes im Sektor gespeichert werden. Des Rätsels Lösung liegt in einer speziellen Codierung, da die Speicherbytes nicht einfach so auf die Diskette geschrieben werden können. Auf diese Weise werden aus 256 Bytes im RAM die 342 Bytes des Diskettensektors. Beim Lesen müssen sie wieder in 256 RAM-Bytes zurückverwandelt werden.
Sowohl am Ende des Daten- als auch des Adressfeldes liegt eine Prüfsumme, mit der kontrolliert werden kann, ob die Bytes des Feldes korrekt gelesen wurden. Sie entsteht dadurch, daß alle Bytes eines Feldes beim Schreiben miteinander in einer Kette EOR-verknüpft werden.

Die Zeit, in der 1 Byte auf die Diskette geschrieben wird, beträgt 32 Microsekunden oder 4 Prozessortakte pro Bit. Da die Umdrehungsgeschwindigkeit einer Diskette nie so konstant ist, daß auf ein Bit genau geschrieben oder gelesen werden kann, sind die Sektoren „schwimmend" in die Sync-Bytes eingelagert, die 3 unterscheidbare Typen von Zwischenräumen (GAP) bilden. Der größte Zwischenraum (GAP1) liegt zwischen Anfang und Ende einer Spur, da sich der erste und letzte Sektor nicht überlappen dürfen. Wir finden hier typischerweise ca. 70 Syncs. Zwischen den übrigen Sektoren liegt GAP3 mit ca. 20 Syncs. Ein weiterer kleiner Zwischenraum (GAP2) liegt zwischen dem Adress- und Datenfeld eines Sektors. Er umfaßt normal 5-8 Syncs.

Abb. 12: Aufbau eines Tracks

Die absolute Länge des GAPs ist nicht konstant, da das Formatierprogramm sie an die Geschwindigkeit des Laufwerks anpaßt. Ein langsames Laufwerk hat längere GAPs als ein schnelles. Zum zweiten wird der Umfang eines Tracks immer geringer, je weiter innen der Track auf der Diskette liegt. Auch diese Unterschiede werden mit den GAPs aufgefangen. Die Formatierprogramme der Firma Apple sind immer bemüht, die Sektoren gleichmäßig über den Umfang des Tracks zu verteilen. Einige Schnellkopierer machen sich diese Mühe nicht. Sie wählen die GAPs 2 und 3 so kurz, das es auf einem schnellen Drive und der innersten Spur noch reicht. Der Ausgleich findet nur über GAP1 statt, während z.B. DOS sowohl GAP1 als auch GAP3 für jede Diskette einzeln anpaßt.

	DOS 3.3	Locksmith 5.0	Superquick	Byte-Blizzard
GAP1	73	107	119	71
GAP2	7	8	8	7
GAP3	19	16	15	20

Sync-Bytes auf Track 0 (Mittelwerte aus 3 Messungen, DOS 3.3 = COPYA)

Die ungleichmäßige Verteilung der Sektoren führt allerdings nach meinen Untersuchungen zu keinen meßbaren Geschwindigkeitsproblemen beim Lesen oder Schreiben (warum Locksmith-kopierte Disketten sich langsamer lesen, folgt später).

Der Datenaustausch zwischen Rechner und Laufwerk führt über ein Datenregister auf dem Controller. Der normale Weg, einen Sektor zu lesen, besteht darin, den Kopf auf die gewünschte Spur zu positionieren und solange das Datenregister zu lesen, bis das gesuchte Adressfeld vorbeikommt. Das unmittelbar folgende Datenfeld wird dann ins RAM gelesen und wieder decodiert. DOS liest immer einen ganzen Sektor auf einmal, ProDOS sogar 2 Sektoren (= 1 Block). Geschrieben wird ganz ähnlich. Zunächst wird solange das Datenregister gelesen, bis das gewünschte Adressfeld vorbeikommt. danach wird sofort auf „Schreiben" umgestellt und das alte Datenfeld mit neuen Daten überschrieben.

Zwischen dem Prozessor des Apple und dem Datenregister werden die Informationen byteweise (= parallel) ausgetauscht. Vom Controller werden die Daten dann bitweise (= seriell) auf die Diskette geschrieben, alle 4 Mikrosekunde 1 Bit. Beim Schreiben muß der Prozessor dem Datenregister exakt alle 32 Mikrosekunde (8*4) ein Byte zur Verfügung stellen, damit der Datenstrom aufrecht erhalten werden kann. Das genaue Timing ist eine der schwierigsten Aufgaben für den Programmierer. Das Lesen ist vergleichsweise einfach: hier wartet der Prozessor einfach ab, bis ein Byte vollständig ist. Dies ist daran zu erkennen, daß Bit 7 des Datenregisters gesetzt ist. Es wird eine Abrufschleife (Polling Loop) von 7 Takten Länge benutzt:

```
LOOP LDA DATENREG
     BPL LOOP
```

Achten Sie darauf, daß nicht zufällig eine Seitengrenze zwischen LOOP und dem Branch liegt, da sonst die Schleife 8 Takte benötigt und unzuverlässig wird!!

Noch eine kleine Anmerkung: Sync-Bytes entstehen, wenn ein $FF dem Datenregister übergeben wird und dann 40 Mikrosekunden bis zum nächsten

$FF vergehen. Dadurch wird die Bitfolge 1111111100 auf die Diskette geschrieben (10-Bit-Byte).

11.2 Von halben Bytes und 6&2

Auf Grund von Hardwarebeschränkungen muß ein Disk-Byte folgende Voraussetzungen erfüllen:
a) Bit 7 muß gesetzt sein
b) maximal zwei aufeinanderfolgende Bits dürfen gelöscht sein
c) von solchen Paaren (b) ist nur eines im Byte erlaubt.

Auf einer normalen 16-Spur-Diskette werden nun zwei Codierverfahren benutzt, um aus Speicher-Bytes die Disk-Bytes werden zu lassen.
Das erste findet sich nur im Adressfeld und wird als 4&4-Codierung bezeichnet. Jedes Speicher-Byte wird einfach in zwei Bytes aufgeteilt und die Lücken mit 1 gefüllt:
Aus ABABABAB wird 1A1A1A1A und 1B1B1B1B.

Der Nachteil dieses Verfahrens ist, daß sich die Byteanzahl verdoppelt und damit die Speicherkapazität der Diskette sinkt. Sein Vorteil ist die Schnelligkeit, in der Codierung und Decodierung erfolgen können. Deshalb wird diese 4&4-Codierung auch für die Adressfelder benutzt.

Die 6&2-Codierung verpackt nun die Speicher-Bytes etwas ökonomischer, aber zugleich auch wesentlich komplizierter. Aus 3 Speicher-Bytes werden dabei nur 4 Disk-Bytes. Von jedem der 3 Speicher-Bytes werden die beiden untersten Bit in einen Zwischenpuffer gespeichert. Die verbliebenen 6 oberen Bits werden zweimal nach rechts geschiftet. Die entstehenden Rest-Bytes haben dadurch Werte zwischen %00000000 und %00111111 ($00–$3F). Die dreimal zwei unteren Bits werden wiederum zu einem Rest-Byte zusammengesetzt. Der Bereich von $00 bis $3F umfaßt 64 verschiedene Möglichkeiten. Zwichen $95 und $FF gibt es 72 gültige Disk-Bytes. Zwei davon ($D5 und $AA) werden reserviert für Epi- und Prolog. Von den restlichen 70 werden nur die 64 benutzt, die wenigsten ein 11-Paar enthalten. Die Umsetzung der Rest-Bytes in die Disk-Bytes geschieht über eine Tabelle. Auf diesem Wege werden aus 256 Speicher-Bytes die 342 Disk-Bytes. Beim Lesen müssen die Disk-Bytes wieder mit einer Tabelle in Rest-Bytes umgesetzt und anschließend zu den Speicher-By-

tes zurückgewandelt werden. Die erforderlichen Programme für beide Richtungen sind recht komplex.

Ein Schritt wurde bisher nicht ganz richtig erklärt: die Reihenfolge, in der die Disk-Bytes auf die Diskette geschrieben werden. Die Codierroutine, auch „Prenibble" genannt, schreibt die beiden unteren Bits jedes Speicher-Bytes in vertauschter Reihenfolge in den Zwischenspeicher. Zusätzlich werden die oberen 6 Bits von unten nach oben im Speicher gelesen, die unteren 2 Bits aber von oben nach unten im Zwischenspeicher abgelegt. Die 256 Speicher-Bytes werden gedrittelt und die Rest-Bytes im Zusatzpuffer werden zu je 2 Bits aus jedem Drittel zusammengesetzt. Das hört sich ungeheuer kompliziert an, ist aber in Assembler relativ einfach zu verwirklichen.

```
AAAAAAam        00AAAAAA        00xlthpd
BBBBBBbn        00BBBBBB        00wksgoc
CCCCCCco        00CCCCCC        00vjrfnb
DDDDDDdp        00DDDDDD        00uipema
EEEEEEeq        00EEEEEE
FFFFFFfr        00FFFFFF
GGGGGGgs        00GGGGGG
HHHHHHht        00HHHHHH
IIIIIIiu        00IIIIII
JJJJJJjv        00JJJJJJ
KKKKKKkw        00KKKKKK
LLLLLLlx        00LLLLLL

Speicher-Bytes  Rest-Bytes      Rest-Bytes
                Hauptpuffer     Zusatzpuffer
```

Da sich 256 nicht ohne Rest durch 3 teilen läßt, wird das letzte Drittel um zwei ungültige Bytes ergänzt. Wir erhalten 256 Bytes im Hauptpuffer und 86 Bytes im Zusatzpuffer. Auf die Diskette werden zunächst die Bytes aus dem Zusatzpuffer geschrieben und dann die Bytes des Hauptpuffers. Als 343. Byte schließt sich die Prüfsumme an, die sich aus der EOR-Verknüpfung der voranstehenden 342 Bytes ergibt.

Der ganze Codierprozess ist sehr zeitaufwendig. Es ist nicht möglich, alle 32 Prozessortakte ein Byte „fertig" zu haben. Bei DOS und ProDOS wird deshalb das Codieren eines Sektors vor dem Schreiben vorgenommen. Es ist nicht möglich, einen Track bei einer Diskettenumdrehung zu beschreiben, weil die Pausen zwischen den Sektoren für die Codierung nicht ausreichen. DOS liest einen Sektor in einen Zwischerpuffer und decodiert ihn nachträglich. Deshalb wird auch hier ein Track nicht in einer Umdrehung gelesen. ProDOS ist dazu in der Lage, da die Decodierung durch viele Tabellen direkt während des Lesens passiert.

Bei einem schnellen Kopierprogramm gibt es zwei Lösungen:
a) Die Disk-Bytes werden gar nicht decodiert, sondern im RAM zwischengespeichert und von dort unverändert herausgeschrieben.
Diese Lösung ist einfach, verbraucht aber viel Speicher.
b) Die Disk-Bytes werden decodiert, aber nach einem anderen Verfahren. Aus dem RAM werden sie mit der Gegenmethode wieder zurückcodiert. Diese Lösung ist platzsparend, aber kompliziert.

Unser Kopierprogramm verwendet die Methode b), die von Omega Ltd. für Locksmith „erfunden" wurde. Der zeitkritische Teil ist das Codieren, nicht das Decodieren. Das Codieren braucht viel Zeit, weil erst der Zusatzpuffer aufgebaut werden muß, da er zuerst geschrieben wird. Die Lösung steckt in folgender Überlegung: Ein Kopierprogramm muß im RAM gar nicht gültige Speicher-Bytes erzeugen. Es ist nur wichtig, daß hinterher auf der Kopie die selben Disk-Bytes stehen. Ein Sektor wird deshalb in Vierergruppen gelesen. Die 6 gültigen Bits des ersten Bytes werden einfach in die 2 freien Bits der nachfolgenden 3 Bytes gepackt. Dadurch entstehen 258 Bytes (3*86). Die 2 „übrigen" Bytes werden in einen Zusatzpuffer geschrieben, da sonst jeder Sektor mehr als eine Speicherseite einnehmen würde. Beim Schreiben der Kopie wird zunächst das 1. Disk-Byte aus den 3 ersten Bytes im RAM wieder herausgelöst. Während dann diese 3 Bytes geschrieben werden, ist genug Zeit, das 4. Disk-Byte aus den nächsten 3 Bytes herauszulösen usw.
Auf diese Weise ist es möglich, einen Track in einer Umdrehung zu lesen und auch zu schreiben.

11.3 Viele kleine Schalterchen

Die Softswitches des Apple haben wir früher schon einmal kennengelernt. Auch der Disk-Controller besitzt 16 solcher Adressen, die An- und Ausschalter für 8 Funktionen darstellen. Sie liegen im Adressbereich $C080,X bis $C08F,X, wobei das X-Register den 16-fachen Wert der Slotnummer enthält. Gerade Adressen schalten „AUS", ungerade „EIN".

$C080,X Phase 0 aus
$C081,X Phase 0 ein
$C082,X Phase 1 aus
$C083,X Phase 1 ein

$C084,X Phase 2 aus
$C085,X Phase 2 ein
$C086,X Phase 3 aus
$C087,X Phase 3 ein

$C088,X Motor aus
$C089,X Motor ein

$C08A,X Laufwerk 1 anwählen
$C08B,X Laufwerk 2 anwählen

$C08C,X Datenregister übertragen
$C08D,X Datenregister laden

$C08E,X Lese-Modus setzen
$C08F,X Schreib-Modus setzen

Die Phasen gehören zum Steppermotor des Kopfes. Wenn die Phasen in aufsteigender Reihenfolge angesprochen werden, bewegt sich der Kopf nach innen. Zur Bewegung von einem Track zum nächsten müssen 2 Phasen geschaltet werden. Es existieren also 70 Positionen bei einer 35-Spur Diskette, die sogenannten „Half-Tracks", die aber nur für Kopierschutzmöglichkeiten eine Rolle spielen. Benutzt werden nur die Positionen, die zu Phase 0 und Phase 2 gehören. Zu Track 0 gehört auch die Phase 0. Für einen runden Motorlauf müssen definierte Anlauf- und Abschaltzeiten eingehalten werden. Nach einem Spurwechsel muß dem Kopf eine Pause zum Auspendeln gegeben werden.

Die Laufwerksanwahl schaltet das Laufwerk nicht an, sondern wählt es nur für die nächste Benutzung vor.
Die Adressen $C08C bis $C08F geben zusammen die Möglichkeit zu vier weiteren Tätigkeiten:
$C08C mit $C08E: Datenregister lesen
$C08D mit $C08E: Schreibschutz testen und Controller initialisieren
$C08C mit $C08F: Datenregister alle 4 Takte um 1 Bit weiterschieben
$C08D mit $C08F: Datenregister beladen

Beispiele:

Laufwerk anwählen

```
LDX #$60      ; Slot 6
LDA $C08A,X   ; Laufwerk 1
   .
LDA $C08B,X   ; Laufwerk 2
```

Motor an/aus

```
LDX #$60      ; Slot 6
LDA $C088,X   ; Motor aus
   .
LDA $C089,X   ; Motor an
```

1 oder mehrere Bytes lesen

```
     LDY #$60        ;Slot 6
     LDA $C08E,X     ;Lese-Modus setzen
     LDA $C08C,X     ;Datenregister normalisieren (1 Zugriff)
LOOP LDA $C08C,X     ;Datenregister laden
     BPL LOOP        ;bei MINUS ist Byte da
      ..
LP2  LDA $C08C,X     ;siehe oben für jedes
     BPL LP2         ;weitere Byte
```

Schreibschutz prüfen, Controller initialisieren

```
LDX #$60       ;Slot 6
LDA $C08D,X
LDA $C08E,X    ;Bit 7 gesetzt, wenn schreibgeschützt
BMI ERROR
 ..
```

Schreiben mehrerer Bytes (vorher Controller initialisieren)

```
LDX #$60       ;Slot 6
LDA BYTE1      ;1. Byte
STA $C08F,X    ;Schreibmodus setzen
CMP $C08C,X    ;schreiben
...
...            ;Wartezyklen einlegen
LDA BYTE2      ;2. Byte
STA $C08D,X    ;Datenregister laden
ORA $C08C,X    ;schreiben
...
...            ;Wartezyklen einlegen
LDA BYTEn      ;n. Byte
STA $C08D,X    ;Datenregister laden
EOR $C08C,X    ;schreiben
...
...            ;Wartezyklen einlegen
LDA $C08E,X    ;Lesemodus setzen
LDA $C08C,X    ;Datenregister normalisieren
```

Das Schreiben von Bytes ist sehr kritisch. Mit einem Test des Schreibschutzes muß zunächst der Controller initialisiert werden. Sodann wird das erste Byte in den Akkumulator geladen und die Adresse $C08F,X angesprochen. Dadurch wird sowohl der Schreibmodus gesetzt als auch das Datenregister geladen. Mit einem Zugriff auf $C08C,X wird dieses Byte dann „abgeschickt". Ab jetzt läuft die Zeit. Das Laden des Datenregisters und das „Abschicken" des Bytes muß mit der indizierten Adressierung erfolgen. *Ein absolutes Adressieren (z.B. STA $C0EC) funktioniert nicht!* Welchen Befehl Sie zum „Abschicken" nehmen, ist relativ gleichgültig: CMP, ORA oder EOR sind gleichermaßen geeignet und benötigen jeweils 4 Takte. STA ist nicht geeignet. Nach genau 32 Prozessortakten muß das Datenregister wieder geladen sein und sofort danach abgeschickt werden. Die Takte zählen vom Beginn des Abschickens bis zum erneuten Abschicken. In dem Programm „Byte-Blizzard" sind an einigen Stellen die Takte mitgezählt, damit Sie eine Vorstellung von den Möglichkeiten haben. Achten Sie darauf, daß keine Seitenübergänge bei einigen Befehlen passieren: LDA $1FF0,X mit X=5 benötigt 4 Takte, mit X=$20 passiert ein Seitenübergang und es werden 5 Takte verbraucht. Außer dem 1. Byte werden alle übrigen mit der Kombination $C08D/$C08C geschrieben.
Warten Sie nach dem Schreiben des letzten Bytes auch 32 Takte, bevor Sie den Lesemodus (LDA $C08E,X) wieder herstellen. Wenn Sie hier nicht lange genug warten, wird Ihr Byte nicht vollständig geschrieben. Der Programmierer von DOS 3.3 hat z.B. eine zu kurze Pause gemacht, als er das Epilog-Byte $EB schreiben mußte. Es ist deshalb immer unvollständig und nicht korrekt lesbar. Warten Sie allerdings mit dem Setzen des Lesemodus zu lange, schreibt der Controller aus dem leeren Datenregister lauter Nullen auf die Diskette. In einem Sekundenbruchteil ist damit ein ganzer Track vernichtet!

Warnung: Alle Programme, die das Diskettenlaufwerk direkt ansprechen, können bei Programmierfehlern die eingelegte Diskette ruinieren. Probieren Sie deshalb solche Programme immer mit Disketten, die auch gelöcht werden dürfen, in allen(!) Laufwerken aus. Es ist ferner ratsam, beim ersten Start jedes Assemblerprogramms die Laufwerksklappen zu öffnen. Sicher ist Sicher!

11.4 Das wirbelnde Byte: Byte-Blizzard

Das Kopierprogramm „Byte-Blizzard" kopiert eine normale 35-Track Apple-Diskette auf einem Apple mit mindestens 64K Hauptspeicher in knapp 32 Sekunden inclusive Formatierung. Da die Formatierung für alle Betriebssy-

steme des Apple (DOS3.3, ProDOS, PASCAL, CP/M) identisch ist und alle Sektoren übertragen werden, kann „Byte-Blizzard" Disketten von allen 4 Betriebssystemen kopieren. Jede geschriebene Spur wird zurückgelesen (Verify). In der folgenden Tabelle sind die Kopierzeiten für einige gebräuchliche Programme zusammengestellt. Bedingungen: 64K Apple IIe mit 2 Laufwerken DISK II am selben Controller in Slot 6, Mittelwerte aus drei handgestoppten Läufen inclusive Formatierung.

Programm	Kopierzeit	Verfahren	„Lesegeschw."
1. Locksmith 5.0	26,6 Sek.	direkt 1 Pass	23,4 Sek.
2. Superquick	30,5 Sek.	direkt 1 Pass	17,2 Sek.
3. Byte-Blizzard	31,8 Sek.	direkt 1 Pass	17,3 Sek.
4. COPY IIplus 5.5	41,9 Sek.	direkt 1 Pass	20,4 Sek.
5. COPY IIE „Stiehl"	65,1 Sek.	RWTS 2 Pass	17,3 Sek.
6. COPYA	98,6 Sek.	RWTS 2 Pass	17,3 Sek.

Beim Testen verschiedener Kopierprogramme fiel mir auf, daß sich die Kopien unterschiedlich gut lesen ließen. Um zu meßbaren Ergebnissen zu kommen, benutzte ich ein Assembler-Programm, daß alle Tracks von 0 bis 34 ($22) einliest. Auf jedem Track wird zunächst Sektor 0 und dann fortlaufend bis Sektor 15 ($F) gelesen, bevor zum nächsten Track gegangen wird. Es wird die Zeit gestoppt, die vom Lesen des ersten bis zum letzten Sektor der Diskette vergeht. Die Ergebnisse finden Sie in der obigen Tabelle unter „Lesegeschwindigkeit". Hier wird ganz deutlich, daß Locksmith aus dem Rahmen fällt und auch COPY IIplus-Produkte noch merklich langsamer gelesen werden als die übrigen Kopien. Zunächst hatte ich die pauschale GAP-Verteilung von Locksmith (siehe 11.1) in Verdacht, aber dann hätten sich auch Superquick-Kopien schlecht lesen lassen müssen.
Des Rätsels Lösung fand sich schließlich in der „Sync-Signature".
Damit ist der Versatz der Tracks gegeneinander gemeint. Wenn Sie Sektor $0 eines Tracks gelesen haben, gehen Sie sofort einen Track weiter nach innen und lesen den nächsten vorbeikommenden Sektor. Je nach Track-Versatz (Sync-Signature) kann das ein beliebiger Sektor von $0 bis $F sein. Es läßt sich leicht zeigen, daß Disketten am schnellsten gelesen werden können, wenn ihre Tracks gegeneinander um -2 versetzt sind, d.h. nach Sektor $0 wird als nächstes der Sektor $E gelesen. Auch ein Wert von -3 ist fast genauso gut (Sektor $D).
Den Zusammenhang zwischen Sync-Signatur und Lesegeschwindigkeit finden Sie in der folgenden Grafik:

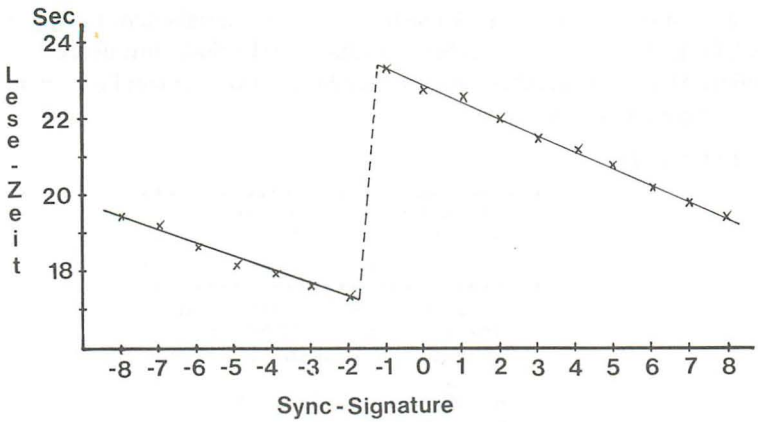


Abb. 13: Lesezeit pro Diskette in Abhängigkeit von der Sync-Signature

Die Kopier- und Formatierprogramme benutzen folgende Sync-Signature:

Programm	Sync-Signature
DOS 3.3	−3
ProDOS 1.0.2	−2
CP/M 2.2	−3
USCD PASCAL	−3
COPYA	−3
COPY IIE	−3
Byte-Blizzard	−3
Superquick	−3 (z.T. −2)
COPY IIplus	+7
Locksmith 5.0	−1

Die Entwickler von Locksmith scheinem diesem Aspekt keine Bedeutung beigemessen zu haben, denn sie wählten den schlechtestmöglichen Wert für ihr Programm aus. Superquick liegt genau zwischen −2 und −3, sodaß kleine Schwankungen der Diskettendrehzahl zu unterschiedlichen Ergebnissen führen, die aber beide gute Leseergebnisse nach sich ziehen.
„Byte-Blizzard" ist meines Wissens der erste Schnellkopierer für den Apple, dessen Quellcode ganz veröffentlicht wird. Der Quellcode ist so umfangreich, daß es nicht mehr mitsamt den Kommentaren in den Assembler ASSESSOR paßt. Benutzen Sie das Hilfsprogramm KOMMEX, um eine „abgespeckte" Version zu erhalten.
Sie können „Byte-Blizzard" als Anregung für Modifikationen nehmen. Wenn Sie mehr oder weniger Speicher haben, ist „Byte-Blizzard" leicht anzupassen.

Auch Disketten mit bis zu 80 Track machen keine Schwierigkeiten. Lediglich 160-Track Disketten mit altenativer Spurumschaltung (Erphi-Controller) erfordern größere Maßnahmen. Aber ansich sollten Sie jetzt vor keinem Problem in Assembler mehr zurückschrecken.

BYTE-BLIZZARD

```
1   ;*******************************
2   ;  Byte-Blizzard  Version 1.0  *
3   ;  von   Dr. Jürgen B. Kehrel  *
4   ;        Dezember  1985        *
5   ;*******************************
6   ; Schnelle Lese- (READ16) und
7   ; Schreibroutinen (WRTRK) in
8   ; Anlehnung an Locksmith 5.0
9   ;
10  ; RDADR16 und SEEKABS sowie
11  ; SUET und LUET nach DOS 3.3
12  ;
13  ;
14  ; Zero-Page Speicherstellen
15  ;
16  SLOT1     EQU  $0006
17  DRIVE1    EQU  $0007
18  SLOT2     EQU  $0008
19  DRIVE2    EQU  $0009
20  TEMP      EQU  $0017
21  INDEX     EQU  $0018
22  CHKSUM    EQU  $0019
23  SEKTOR    EQU  $001E
24  SYNCS     EQU  $001F
25  CH        EQU  $0024
26  PTR       EQU  $00CE       ;+$00CF
27  SPURIST   EQU  $00EB
28  SPURSOLL  EQU  $00EC
29  ;
30  ; Reset-Vektor, Bildschirm
31  ;
32  SOFTEV    EQU  $03F2
33  ZEILE14   EQU  $072A
34  ;
35  ; Tabellen zum Lesen
36  ;
37  RDTRANS1  EQU  $0300
38  RDTRANS2  EQU  $0340
39  RDTRANS3  EQU  $0380
40  ;
41  ; Tabellen zum Schreiben
42  ;
43  WRTRANS1  EQU  $0800
44  WRTRANS2  EQU  $0900
45  WRTRANS3  EQU  $0A00
46  WRTRANS4  EQU  $0B00
47  ;
48  ; Beginn des Datenpuffers
49  ;
```

```
50   PUFFER   EQU  $2000
51   ;
52   ; Tastatur
53   ;
54   KEY      EQU  $C000
55   STROBE   EQU  $C010
56   ;
57   ; Apple //e,c 80-Zeichenkarte
58   ;
59   STR80FF EQU  $C000
60   COL80FF EQU  $C00C
61   ;
62   ; Softswitches
63   ;
64   ROM      EQU  $C082
65   LCRAMB2  EQU  $C083
66   ;
67   ; Grundadressen Disk-Controller
68   ;
69   PHASE0F EQU  $C080
70   MOTOROFF EQU  $C088
71   MOTORON  EQU  $C089
72   DRV1ENAB EQU  $C08A
73   DRV2ENAB EQU  $C08B
74   DATASTRB EQU  $C08C
75   LOADDATA EQU  $C08D
76   INPUTMOD EQU  $C08E
77   OUTMODUS EQU  $C08F
78   DSTRBSL6 EQU  $C0EC
79   LDATASL6 EQU  $C0ED
80   INPMDSL6 EQU  $C0EE
81   ;
82   ; RAM-Karte
83   ;
84   CARDOFF  EQU  $CFFF
85   BANK     EQU  $D000
86   ;
87   ; Monitor-ROM
88   ;
89   TEXT     EQU  $FB2F
90   TABV     EQU  $FB5B
91   BELL     EQU  $FBDD
92   HOME     EQU  $FC58
93   CLREOL   EQU  $FC9C
94   RDKEY    EQU  $FD0C
95   CROUT    EQU  $FD8E
96   COUT1    EQU  $FDF0
97   SETINV   EQU  $FE80
98   SETNORM  EQU  $FE84
99   SETKBD   EQU  $FE89
100  SETVID   EQU  $FE93
101  RESET    EQU  $FFFC
102  ;
103  ;
```

```
                        104  ; $0300 - $03BF und $0800 - $0BFF
                        105  ; mit Tabellen belegt, die vom
                        106  ; Programm erzeugt werden.
                        107  ;
                        108          ORG  $0C00
                        109  ;
                        110  ;
0C00: 4C 00 10          111  KALTSTRT JMP  START
                        112  ;
                        113  ; Schreiben eines Tracks
                        114  ;
0C03: A6 08             115  WRTRK    LDX  SLOT2       ;Slot Copy
0C05: BD 8D C0          116           LDA  LOADDATA,X
0C08: BD 8E C0          117           LDA  INPUTMOD,X  ;Write-Protected?
0C0B: 10 06             118           BPL  WRSYNC      ;Nein, weiter
0C0D: BD 8C C0          119           LDA  DATASTRB,X  ;Ja
0C10: 4C 19 16          120           JMP  WPROT       ;zurück
                        121  ;
                        122  ; viele Sync-Bytes schreiben ; GAP 1
                        123  ;
0C13: A9 FF             124  WRSYNC   LDA  #$FF
0C15: 9D 8F C0          125           STA  OUTMODUS,X  ;Write/Load
0C18: 1D 8C C0          126           ORA  DATASTRB,X  ;Latch
0C1B: A0 70             127           LDY  #$70
0C1D: D0 00             128           BNE  SYNCLOOP
0C1F: 20 E4 0D          129  SYNCLOOP JSR  WAIT27      ;36 Takte
0C22: 8D ED C0          130  WRS1     STA  LDATASL6    ;40
0C25: 0D EC C0          131  WRS2     ORA  DSTRBSL6    ;schreiben 4
0C28: 88                132           DEY              ;6
0C29: D0 F4             133           BNE  SYNCLOOP    ;9 (8)
                        134  ;
0C2B: 20 E7 0D          135           JSR  WAIT23      ;31
                        136  ;
                        137  ; Seiteneinstieg für GAP 3
                        138  ; Adressen modifiz. je nach Sektor
                        139  ;
0C2E: 84 1E             140  SYNCBYTE STY  SEKTOR      ;34
0C30: A9 FF             141           LDA  #$FF        ;36
0C32: 8D ED C0          142  WRS3     STA  LDATASL6    ;40
0C35: 0D EC C0          143  WRS4     ORA  DSTRBSL6    ;schreiben
0C38: 98                144           TYA              ;6
0C39: 0D 32 0F          145           ORA  STORE       ;10
0C3C: AA                146           TAX              ;12
0C3D: A9 FF             147           LDA  #$FF        ;14
0C3F: EA                148           NOP              ;16
0C40: 8E 6F 0C          149           STX  SYB1+2      ;20
0C43: 8E 75 0C          150           STX  SYB2+2      ;24
0C46: 8E 7B 0C          151           STX  SYB3+2      ;28
0C49: 8E 19 0D          152           STX  ZPL1+2      ;32
0C4C: 8E 28 0D          153           STX  ZPL2+2      ;36
0C4F: 8D ED C0          154  WRS5     STA  LDATASL6    ;40
0C52: 0D EC C0          155  WRS6     ORA  DSTRBSL6    ;4
0C55: 8E 31 0D          156           STX  ZPL3+2      ;8
0C58: 8E 40 0D          157           STX  ZPL4+2      ;12
```

```
0C5B: 8E 49 0D  158           STX  ZPL5+2      ;16
0C5E: 8E 5B 0D  159           STX  ZPL6+2      ;20
0C61: 8E 71 0D  160           STX  LAST+2      ;24
0C64: 20 EC 0D  161           JSR  WAITRTS     ;36
0C67: 8D ED C0  162 WRS7      STA  LDATASL6    ;40
0C6A: 0D EC C0  163 WRS8      ORA  DSTRBSL6    ;4
                164 ;
                165 ; 1. Byte zum Schreiben vorbereiten
                166 ;
0C6D: AE 54 20  167 SYB1      LDX  PUFFER+$54 ;8
0C70: BD 00 08  168           LDA  WRTRANS1,X ;12
0C73: AE A9 20  169 SYB2      LDX  PUFFER+$A9 ;16
0C76: 1D 00 09  170           ORA  WRTRANS2,X ;20
0C79: AE FE 20  171 SYB3      LDX  PUFFER+$FE ;24
0C7C: 1D 00 0A  172           ORA  WRTRANS3,X ;28
0C7F: 8D 31 0F  173           STA  NIBBTEMP   ;32
0C82: EA        174           NOP             ;34
0C83: A9 FF     175           LDA  #$FF       ;36
0C85: 8D ED C0  176 WRS9      STA  LDATASL6   ;40
0C88: 0D EC C0  177 WRS10     ORA  DSTRBSL6   ;4
                178 ;
                179 ; Spurspezifische Anzahl von
                180 ; Synchronisations-Bytes
                181 ;
0C8B: A4 1F     182           LDY  SYNCS      ;Anzahl (7)
0C8D: EA        183           NOP             ;9
0C8E: 20 E4 0D  184 GAP1SYNC  JSR  WAIT27     ;36
0C91: 8D ED C0  185 WRS11     STA  LDATASL6   ;40
0C94: 0D EC C0  186 WRS12     ORA  DSTRBSL6   ;4
0C97: 88        187           DEY             ;6
0C98: D0 F4     188           BNE  GAP1SYNC   ;9 (8)
                189 ;
                190 ; Adressfeld-Prolog schreiben
                191 ;
0C9A: A9 D5     192           LDA  #$D5       ;10
0C9C: 20 D7 0D  193           JSR  WBYTE16    ;letztes FF kein Sync
0C9F: A9 AA     194           LDA  #$AA       ;12
0CA1: 20 D8 0D  195           JSR  WBYTE14    ;18+14=32
0CA4: A9 96     196           LDA  #$96       ;12
0CA6: 20 D8 0D  197           JSR  WBYTE14    ;18+14=32
                198 ;
                199 ; Adress-Bytes (VOL, TRK, SEC, CHK)
                200 ;
0CA9: AD 3B 0F  201           LDA  RVOL       ;14
0CAC: 20 BD 0D  202           JSR  WRODDEVE   ;20+12=32
0CAF: AD 36 0F  203           LDA  TRACKC     ;14
0CB2: 20 BD 0D  204           JSR  WRODDEVE   ;20+12=32
0CB5: 85 17     205           STA  TEMP       ;13 Anpassung
0CB7: A5 1E     206           LDA  SEKTOR     ;16
0CB9: 20 BE 0D  207           JSR  WRODDEV1   ;22+10=32
0CBC: AD 3B 0F  208           LDA  RVOL       ;14
0CBF: 4D 36 0F  209           EOR  TRACKC     ;18
0CC2: 45 1E     210           EOR  SEKTOR     ;21
0CC4: 48        211           PHA             ;24 merken
```

```
0CC5: 4A          212           LSR             ;26
0CC6: 09 AA       213           ORA  #$AA       ;28
0CC8: 8D ED CO    214  WRS13     STA  LDATASL6   ;32
0CCB: 0D EC CO    215  WRS14     ORA  DSTRBSL6   ;4
0CCE: 68          216           PLA             ;8  zurück
0CCF: 09 AA       217           ORA  #$AA       ;10
0CD1: 20 D7 0D    218           JSR  WBYTE16    ;16+16=32
                  219  ;
                  220  ; Adressfeld-Epilog schreiben
                  221  ;
0CD4: A9 DE       222           LDA  #$DE       ;12
0CD6: 20 D8 0D    223           JSR  WBYTE14    ;18+14=32
0CD9: A9 AA       224           LDA  #$AA       ;12
0CDB: 20 D8 0D    225           JSR  WBYTE14    ;18+14=32
0CDE: A9 EB       226           LDA  #$EB       ;12
0CE0: 20 D8 0D    227           JSR  WBYTE14    ;18+14=32
                  228  ;
                  229  ; 7 Sync-Bytes GAP 2
                  230  ;
0CE3: EA          231           NOP             ;12 Takte
0CE4: EA          232           NOP             ;14
0CE5: 48          233           PHA             ;17
0CE6: 68          234           PLA             ;21
0CE7: A0 07       235           LDY  #$07       ;23
0CE9: A9 FF       236           LDA  #$FF       ;25
0CEB: D0 03       237           BNE  GAP2       ;28
                  238  ;
0CED: 20 E4 0D    239  GAP2LOOP JSR  WAIT27
0CF0: 8D ED CO    240  GAP2     STA  LDATASL6   ;(32)
0CF3: 0D EC CO    241  WRS15    ORA  DSTRBSL6
0CF6: 88          242           DEY
0CF7: D0 F4       243           BNE  GAP2LOOP
                  244  ;
                  245  ; Datenfeld-Prolog schreiben
                  246  ;
0CF9: A9 D5       247           LDA  #$D5
0CFB: 20 D3 0D    248           JSR  WBYTE24    ;alle FF's Sync
0CFE: A9 AA       249           LDA  #$AA
0D00: 20 D8 0D    250           JSR  WBYTE14
0D03: A9 AD       251           LDA  #$AD
0D05: 20 D8 0D    252           JSR  WBYTE14
                  253  ;
                  254  ; Die 342 Daten-Nibble schreiben
                  255  ;
0D08: AE 31 0F    256           LDX  NIBBTEMP   ;vorab kodiert
0D0B: BD 00 0B    257           LDA  WRTRANS4,X ;18
0D0E: 20 DB 0D    258           JSR  WBYTE8     ;24+8=32
0D11: A0 54       259           LDY  #$54       ;12
0D13: EA          260           NOP             ;14 Anpassung
                  261  ;
0D14: EE 30 0F    262  ZAPLOOP  INC  DUMMY      ;20 Anpassung
0D17: BE 00 20    263  ZPL1     LDX  PUFFER,Y   ;24
0D1A: BD 00 0B    264           LDA  WRTRANS4,X ;28
0D1D: 8D ED CO    265  WRS16    STA  LDATASL6   ;32
```

```
0D20: 0D EC C0  266  WRS17   ORA  DSTRBSL6    ;4
0D23: AD 31 0F  267          LDA  NIBBTEMP    ;8
0D26: BE A9 20  268  ZPL2    LDX  PUFFER+$A9,Y ;12
0D29: BD 00 0A  269          LDA  WRTRANS3,X ;16
0D2C: 8D 31 0F  270          STA  NIBBTEMP    ;20
0D2F: BE 55 20  271  ZPL3    LDX  PUFFER+$55,Y ;24
0D32: BD 00 0B  272          LDA  WRTRANS4,X ;28
0D35: 8D ED C0  273  WRS18   STA  LDATASL6    ;32
0D38: 0D EC C0  274  WRS19   ORA  DSTRBSL6    ;4
0D3B: AD 31 0F  275          LDA  NIBBTEMP    ;8
0D3E: BE 54 20  276  ZPL4    LDX  PUFFER+$54,Y ;12
0D41: 1D 00 09  277          ORA  WRTRANS2,X ;16
0D44: 8D 31 0F  278          STA  NIBBTEMP    ;20
0D47: BE AA 20  279  ZPL5    LDX  PUFFER+$AA,Y ;24
0D4A: BD 00 0B  280          LDA  WRTRANS4,X ;28
0D4D: 8D ED C0  281  WRS20   STA  LDATASL6    ;32
0D50: 0D EC C0  282  WRS21   ORA  DSTRBSL6    ;4
0D53: 88        283          DEY              ;6
0D54: 30 19     284          BMI  LAST        ;8 (9)
0D56: AD 31 0F  285          LDA  NIBBTEMP    ;12
0D59: BE 00 20  286  ZPL6    LDX  PUFFER,Y    ;16
0D5C: 1D 00 08  287          ORA  WRTRANS1,X ;20
0D5F: AA        288          TAX              ;22
0D60: EA        289          NOP              ;24 Anpassung
0D61: BD 00 0B  290          LDA  WRTRANS4,X ;28
0D64: 8D ED C0  291  WRS22   STA  LDATASL6    ;32
0D67: 0D EC C0  292  WRS23   ORA  DSTRBSL6    ;4
0D6A: 48        293          PHA              ;7 Anpassung
0D6B: 68        294          PLA              ;11   dito
0D6C: 4C 14 0D  295          JMP  ZAPLOOP     ;14
                296  ;
0D6F: AE FF 20  297  LAST    LDX  PUFFER+$FF ;13
0D72: BD 00 0B  298          LDA  WRTRANS4,X ;17
0D75: 20 D0 0D  299          JSR  WBYTE9      ;23+9=32
0D78: A5 1E     300          LDA  SEKTOR      ;13
0D7A: 0A        301          ASL              ;15
0D7B: A8        302          TAY              ;17
0D7C: B1 CE     303          LDA  (PTR),Y     ;1. Extrabyte (22)
0D7E: AA        304          TAX              ;24
0D7F: BD 00 0B  305          LDA  WRTRANS4,X ;28
0D82: 8D ED C0  306  WRS24   STA  LDATASL6    ;32
0D85: 0D EC C0  307  WRS25   ORA  DSTRBSL6    ;4
0D88: C8        308          INY              ;6
0D89: B1 CE     309          LDA  (PTR),Y     ;2. Extrabyte (11
0D8B: AA        310          TAX              ;13
0D8C: BD 00 0B  311          LDA  WRTRANS4,X ;17
0D8F: 20 D0 0D  312          JSR  WBYTE9      ;23+9=32
                313  ;
                314  ; Datenfeld-Epilog schreiben
                315  ;
0D92: A9 DE     316          LDA  #$DE        ;12
0D94: 20 D8 0D  317          JSR  WBYTE14     ;18+14=32
0D97: A9 AA     318          LDA  #$AA        ;12
0D99: 20 D8 0D  319          JSR  WBYTE14
```

```
0D9C: A9 EB     320          LDA  #$EB
0D9E: 20 D8 0D  321          JSR  WBYTE14
0DA1: A9 FF     322          LDA  #$FF
0DA3: 20 D8 0D  323          JSR  WBYTE14    ;Übergang
                324  ;
                325  ; Prüfen, ob alle Sektoren geschrieben
                326  ;
0DA6: EE 30 0F  327          INC  DUMMY      ;16
0DA9: A4 1E     328          LDY  SEKTOR     ;19
0DAB: C8        329          INY             ;21
0DAC: C0 10     330          CPY  #$10       ;23
0DAE: B0 05     331          BCS  TRKREADY   ;25 (26)
0DB0: 84 1E     332          STY  SEKTOR     ;28
0DB2: 4C 2E 0C  333          JMP  SYNCBYTE   ;31
                334  ;
                335  ; Trackfertig, Read, Schleuse normal.
                336  ;
0DB5: EA        337  TRKREADY NOP
0DB6: AD EE C0  338  WRS26    LDA  INPMDSL6
0DB9: 0D EC C0  339  WRS27    ORA  DSTRBSL6
0DBC: 60        340          RTS
                341  ;
                342  ; Schreibe Byte Odd-Even Codiert
                343  ;
0DBD: EA        344  WRODDEVE NOP
0DBE: AA        345  WRODDEV1 TAX
0DBF: 4A        346          LSR
0DC0: 09 AA     347          ORA  #$AA
0DC2: 8D ED C0  348  WRS28    STA  LDATASL6
0DC5: 0D EC C0  349  WRS29    ORA  DSTRBSL6
0DC8: 8A        350          TXA
0DC9: 09 AA     351          ORA  #$AA
0DCB: 85 17     352          STA  TEMP
0DCD: 4C D6 0D  353          JMP  WBYTE18
                354  ;
                355  ; Zeitverzögerung, dann schreiben
                356  ;
0DD0: 4C DC 0D  357  WBYTE9   JMP  WBYTE6
                358  ;
0DD3: EE 30 0F  359  WBYTE24  INC  DUMMY
0DD6: EA        360  WBYTE18  NOP
0DD7: EA        361  WBYTE16  NOP
0DD8: EE 30 0F  362  WBYTE14  INC  DUMMY
0DDB: EA        363  WBYTE8   NOP
0DDC: EA        364  WBYTE6   NOP
0DDD: 8D ED C0  365  WRS30    STA  LDATASL6
0DE0: 0D EC C0  366  WRS31    ORA  DSTRBSL6
0DE3: 60        367          RTS
                368  ;
                369  ; Zeitverzögerungen
                370  ;
0DE4: 8D 30 0F  371  WAIT27   STA  DUMMY
0DE7: 48        372  WAIT23   PHA
0DE8: 68        373          PLA
```

```
0DE9: 8D 30 0F  374            STA  DUMMY
0DEC: 60        375  WAITRTS   RTS
                376  ;
                377  ;
                378  ; Merkbereich für Sektoren
                379  ;
                380  SIM       DFS  $10
                381  ;
                382  ; Daten lesen, Puffer immer auf
                383  ; Seitengrenze, Hi-Byte wird übergeben
                384  ; X-Reg muß Slot*16 enthalten
                385  ;
0DFD: AD 33 0F  386  READ16    LDA  ZIELHI
0E00: 8D 56 0E  387            STA  RDN1+2
0E03: 8D 6A 0E  388            STA  RDN2+2
0E06: 8D 7E 0E  389            STA  RDN3+2
0E09: 8D C4 0E  390            STA  RDD1+2
                391  ;
                392  ; Datenfeld lesen , 32 Versuche
                393  ;
0E0C: A0 20     394  RDDAPROL LDY  #$20
0E0E: 88        395  RDVERS   DEY
0E0F: 10 02     396            BPL  DAPROL1
0E11: 38        397            SEC
0E12: 60        398            RTS              ;Fehlerausstieg
                399  ;
                400  ; Datenfeld-Prolog suchen
                401  ;
0E13: BD 8C C0  402  DAPROL1   LDA  DATASTRB,X
0E16: 10 FB     403            BPL  DAPROL1
0E18: 49 D5     404  RDPROL2  EOR  #$D5
0E1A: D0 F2     405            BNE  RDVERS
0E1C: EA        406            NOP
0E1D: BD 8C C0  407  RDPROL3  LDA  DATASTRB,X
0E20: 10 FB     408            BPL  RDPROL3
0E22: C9 AA     409            CMP  #$AA
0E24: D0 F2     410            BNE  RDPROL2
0E26: EA        411            NOP
0E27: BD 8C C0  412  RDPROL4  LDA  DATASTRB,X
0E2A: 10 FB     413            BPL  RDPROL4
0E2C: C9 AD     414            CMP  #$AD
0E2E: D0 E8     415            BNE  RDPROL2
                416  ;
                417  ; Disk-Nibble lesen und ablegen
                418  ;
0E30: AD 39 0F  419            LDA  RSEC        ;Index im Zusatz-
0E33: 0A        420            ASL              ;Puffer für die 2
0E34: 85 18     421            STA  INDEX       ;„Extrabytes"
0E36: 24 17     422            BIT  TEMP
0E38: AE EC C0  423  RDNIBBL  LDX  DSTRBSL6
0E3B: 10 FB     424            BPL  RDNIBBL
0E3D: BD 00 0F  425            LDA  LUET-$96,X
0E40: 85 17     426            STA  TEMP
0E42: 85 19     427            STA  CHKSUM
```

```
0E44: A0 54     428           LDY  #$54
0E46: EA        429           NOP
0E47: AE EC C0  430 RDNIBBL1  LDX  DSTRBSL6
0E4A: 10 FB     431           BPL  RDNIBBL1
0E4C: BD 00 0F  432           LDA  LUET-$96,X
0E4F: A6 17     433           LDX  TEMP
0E51: 1D 00 03  434           ORA  RDTRANS1,X
0E54: 99 00 20  435 RDN1      STA  PUFFER,Y
0E57: 45 19     436           EOR  CHKSUM
0E59: 85 19     437           STA  CHKSUM
0E5B: AE EC C0  438 RDNIBBL2  LDX  DSTRBSL6
0E5E: 10 FB     439           BPL  RDNIBBL2
0E60: BD 00 0F  440           LDA  LUET-$96,X
0E63: A6 17     441           LDX  TEMP
0E65: 1D 40 03  442           ORA  RDTRANS2,X
0E68: 99 55 20  443 RDN2      STA  PUFFER+$55,Y
0E6B: 45 19     444           EOR  CHKSUM
0E6D: 85 19     445           STA  CHKSUM
0E6F: AE EC C0  446 RDNIBBL3  LDX  DSTRBSL6
0E72: 10 FB     447           BPL  RDNIBBL3
0E74: BD 00 0F  448           LDA  LUET-$96,X
0E77: A6 17     449           LDX  TEMP
0E79: 1D 80 03  450           ORA  RDTRANS3,X
0E7C: 99 AA 20  451 RDN3      STA  PUFFER+$AA,Y
0E7F: 45 19     452           EOR  CHKSUM
0E81: 85 19     453           STA  CHKSUM
0E83: AE EC C0  454 RDNIBBL4  LDX  DSTRBSL6
0E86: 10 FB     455           BPL  RDNIBBL4
0E88: BD 00 0F  456           LDA  LUET-$96,X
0E8B: 85 17     457           STA  TEMP
0E8D: 45 19     458           EOR  CHKSUM
0E8F: 85 19     459           STA  CHKSUM
0E91: 88        460           DEY
0E92: 10 B3     461           BPL  RDNIBBL1
0E94: 24 17     462           BIT  TEMP
0E96: AE EC C0  463 RDNIBBL5  LDX  DSTRBSL6
0E99: 10 FB     464           BPL  RDNIBBL5
0E9B: BD 00 0F  465           LDA  LUET-$96,X
0E9E: A4 18     466           LDY  INDEX
0EA0: 91 CE     467           STA  (PTR),Y     ;1. Extrabyte
0EA2: 45 19     468           EOR  CHKSUM
0EA4: 29 3F     469           AND  #$3F
0EA6: 85 19     470           STA  CHKSUM
0EA8: AE EC C0  471 RDNIBBL6  LDX  DSTRBSL6
0EAB: 10 FB     472           BPL  RDNIBBL6
0EAD: BD 00 0F  473           LDA  LUET-$96,X
0EB0: C8        474           INY
0EB1: 91 CE     475           STA  (PTR),Y     ;2. Extrabyte
0EB3: C5 19     476           CMP  CHKSUM
0EB5: D0 1B     477           BNE  RDERROR
                478 ;
                479 ; Datenfeld-Epilog lesen
                480 ;
0EB7: AE EC C0  481 RDDAEPIL  LDX  DSTRBSL6
```

```
0EBA: 10 FB      482           BPL  RDDAEPIL
0EBC: E0 DE      483           CPX  #$DE
0EBE: D0 12      484           BNE  RDERROR
0EC0: A5 17      485           LDA  TEMP
0EC2: 8D FF 20   486  RDD1     STA  PUFFER+$FF
0EC5: 24 17      487           BIT  TEMP
0EC7: AE EC C0   488  RDDAEPI1 LDX  DSTRBSL6
0ECA: 10 FB      489           BPL  RDDAEPI1
0ECC: E0 AA      490           CPX  #$AA
0ECE: D0 02      491           BNE  RDERROR
0ED0: 18         492           CLC
0ED1: 60         493           RTS
                 494  ;
0ED2: 38         495  RDERROR  SEC
0ED3: 60         496           RTS
                 497  ;
                 498  ; Adressfeld lesen
                 499  ; Adressfeld-Prolog suchen
                 500  ; X muß Slot*16 enthalten!
                 501  ;
0ED4: A0 FC      502  RDADR16  LDY  #$FC       ;Versuchszahl
0ED6: 84 17      503           STY  TEMP
0ED8: C8         504  VERSUCH  INY
0ED9: D0 04      505           BNE  RAPROL
0EDB: E6 17      506           INC  TEMP
0EDD: F0 F3      507           BEQ  RDERROR
                 508  ;
0EDF: BD 8C C0   509  RAPROL   LDA  DATASTRB,X
0EE2: 10 FB      510           BPL  RAPROL
0EE4: C9 D5      511  RAPROL1  CMP  #$D5
0EE6: D0 F0      512           BNE  VERSUCH
0EE8: EA         513           NOP
0EE9: BD 8C C0   514  RAPROL2  LDA  DATASTRB,X
0EEC: 10 FB      515           BPL  RAPROL2
0EEE: C9 AA      516           CMP  #$AA
0EF0: D0 F2      517           BNE  RAPROL1
0EF2: A0 03      518           LDY  #$03        ;4 Bytes
0EF4: BD 8C C0   519  RAPROL3  LDA  DATASTRB,X
0EF7: 10 FB      520           BPL  RAPROL3
0EF9: C9 96      521           CMP  #$96
0EFB: D0 E7      522           BNE  RAPROL1
                 523  ;
                 524  ; Adressfeld lesen, Bytes zusammensetzen
                 525  ;
0EFD: A9 00      526           LDA  #$00        ;Chksum init.
0EFF: 85 19      527  ADRLOOP  STA  CHKSUM
0F01: BD 8C C0   528  ADR1     LDA  DATASTRB,X
0F04: 10 FB      529           BPL  ADR1
0F06: 2A         530           ROL
0F07: 85 17      531           STA  TEMP
0F09: BD 8C C0   532  ADR2     LDA  DATASTRB,X
0F0C: 10 FB      533           BPL  ADR2
0F0E: 25 17      534           AND  TEMP
0F10: 99 38 0F   535           STA  ADRINFO,Y   ;Werte merken
```

```
0F13: 45 19     536           EOR  CHKSUM
0F15: 88        537           DEY
0F16: 10 E7     538           BPL  ADRLOOP
                539  ;
0F18: A8        540           TAY              ;Chksum muß 0 sein
0F19: D0 B7     541           BNE  RDERROR
                542  ;
                543  ; Adressfeld-Epilog lesen
                544  ;
0F1B: BD 8C C0  545  RAEPIL   LDA  DATASTRB,X
0F1E: 10 FB     546           BPL  RAEPIL
0F20: C9 DE     547           CMP  #$DE
0F22: D0 AE     548           BNE  RDERROR
0F24: EA        549           NOP
0F25: BD 8C C0  550  RAEPI1   LDA  DATASTRB,X
0F28: 10 FB     551           BPL  RAEPI1
0F2A: C9 AA     552           CMP  #$AA
0F2C: D0 A4     553           BNE  RDERROR
0F2E: 18        554           CLC
0F2F: 60        555           RTS
                556  ;
                557  ; Variablen-Liste
                558  ;
0F30: 00        559  DUMMY    HEX  00
0F31: 00        560  NIBBTEMP HEX  00
0F32: 00        561  STORE    HEX  00
0F33: 00        562  ZIELHI   HEX  00
0F34: 00        563  TRACK    HEX  00
0F35: 00        564  TRACK0   HEX  00
0F36: 00        565  TRACKC   HEX  00
0F37: 00        566  TRKPOS   HEX  00
0F38: 00        567  ADRINFO  HEX  00
0F39: 00        568  RSEC     HEX  00
0F3A: 00        569  RTRK     HEX  00
0F3B: 00        570  RVOL     HEX  00
0F3C: 00        571  RDVER    HEX  00
0F3D: 00        572  WRVERS   HEX  00
                573  ;
                574  ;
                575  ; (PTR) auf Merkbereich für 2 Zusatzbytes
                576  ; für jeden gelesenen Sektor zeigen lassen
                577  ; 32 ($20) Bytes pro Track erforderlich
                578  ;
0F3E: A9 00     579  MERK     LDA  #$00
0F40: 85 CF     580           STA  PTR+1       ;Hi-Byte
0F42: AD 34 0F  581           LDA  TRACK       ;Track im Durchgang
0F45: A0 05     582           LDY  #$05        ;; 32
0F47: 0A        583  MLOOP    ASL
0F48: 26 CF     584           ROL  PTR+1       ;Übertrag
0F4A: 88        585           DEY
0F4B: D0 FA     586           BNE  MLOOP
0F4D: 85 CE     587           STA  PTR         ;Lo-Byte
0F4F: A9 1E     588           LDA  #$1E        ;$1E00-$1FFF bei
0F51: 65 CF     589           ADC  PTR+1       ;max. 16 Tracks
```

```
0F53: 85 CF      590           STA  PTR+1
0F55: 60         591           RTS
                 592  ;
                 593  ;
                 594  ; Schreib-Übersetzungstabelle SÜT
                 595  ;
0F56: 96 97 9A   596  SUET     HEX  96979A9B9D9E9FA6
0F59: 9B 9D 9E 9F A6
0F5E: A7 AB AC   597           HEX  A7ABACADAEAFB2B3
0F61: AD AE AF B2 B3
0F66: B4 B5 B6   598           HEX  B4B5B6B7B9BABBBC
0F69: B7 B9 BA BB BC
0F6E: BD BE BF   599           HEX  BDBEBFCBCDCECFD3
0F71: CB CD CE CF D3
0F76: D6 D7 D9   600           HEX  D6D7D9DADBDCDDDE
0F79: DA DB DC DD DE
0F7E: DF E5 E6   601           HEX  DFE5E6E7E9EAEBEC
0F81: E7 E9 EA EB EC
0F86: ED EE EF   602           HEX  EDEEEFF2F3F4F5F6
0F89: F2 F3 F4 F5 F6
0F8E: F7 F9 FA   603           HEX  F7F9FAFBFCFDFEFF
0F91: FB FC FD FE FF
                 604  ;
                 605  ; Lese-Übersetzungstabelle
                 606  ; Ungültige Diskbytes = FF
                 607  ; indiziert zur Basis LUET-$96
                 608  ; muß auf $xx96 beginnen!
                 609  ;
0F96: 00 01 FF   610  LUET     HEX  0001FFFF0203FF04
0F99: FF 02 03 FF 04
0F9E: 05 06 FF   611           HEX  0506FFFFFFFFFFFF
0FA1: FF FF FF FF FF
0FA6: 07 08 FF   612           HEX  0708FFFFFF090A0B
0FA9: FF FF 09 0A 0B
0FAE: 0C 0D FF   613           HEX  0C0DFFFF0E0F1011
0FB1: FF 0E 0F 10 11
0FB6: 12 13 FF   614           HEX  1213FF1415161718
0FB9: 14 15 16 17 18
0FBE: 19 1A FF   615           HEX  191AFFFFFFFFFFFF
0FC1: FF FF FF FF FF
0FC6: FF FF FF   616           HEX  FFFFFFFFFF1BFF1C
0FC9: FF FF 1B FF 1C
0FCE: 1D 1E FF   617           HEX  1D1EFFFFFF1FFFFF
0FD1: FF FF 1F FF FF
0FD6: 20 21 FF   618           HEX  2021FF2223242526
0FD9: 22 23 24 25 26
0FDE: 27 28 FF   619           HEX  2728FFFFFFFFFF29
0FE1: FF FF FF FF 29
0FE6: 2A 2B FF   620           HEX  2A2BFF2C2D2E2F30
0FE9: 2C 2D 2E 2F 30
0FEE: 31 32 FF   621           HEX  3132FFFF33343536
0FF1: FF 33 34 35 36
0FF6: 37 38 FF   622           HEX  3738FF393A3B3C3D
0FF9: 39 3A 3B 3C 3D
```

```
0FFE: 3E 3F      623          HEX  3E3F
                 624  ;
                 625  ;
                 626  ; Initialisieren, Tabellen anlegen
                 627  ;
1000: 8D 00 C0   628  START    STA  STR80OFF
1003: 8D 0C C0   629          STA  C0L80OFF   ;80-Zeichen aus (IIe/c)
1006: 8D FF CF   630          STA  CARDOFF    ;Zusatz-ROMs aus
1009: 20 2F FB   631          JSR  TEXT
100C: 20 BB 17   632          JSR  MAKETABS
                 633  ;
                 634  ; Teste, ob Language Card vorhanden
                 635  ;
100F: AD 83 C0   636          LDA  LCRAMB2    ;RAM-Read Bank 2
1012: AD 83 C0   637          LDA  LCRAMB2    ;RAM-Write Bank 2
1015: AD 00 D0   638          LDA  BANK       ;1. Byte der Karte
1018: 49 FF      639          EOR  #$FF       ;invertieren
101A: 8D 00 D0   640          STA  BANK       ;zurückschreiben
101D: CD 00 D0   641          CMP  BANK       ;ist es verändert?
1020: D0 13      642          BNE  NOTLC      ;keine Karte da
1022: A9 55      643          LDA  #$55       ;%0101 0101
1024: 8D 00 D0   644          STA  BANK
1027: CD 00 D0   645          CMP  BANK       ;wieder gleich?
102A: D0 09      646          BNE  NOTLC
102C: 0A         647          ASL
102D: 0E 00 D0   648          ASL  BANK
1030: 4D 00 D0   649          EOR  BANK
1033: F0 06      650          BEQ  LCOK       ;Ja, also Karte da
1035: AD 82 C0   651  NOTLC    LDA  ROM        ;ROMs ein
1038: 4C D4 15   652          JMP  LCERROR    ;Fehler, keine LC da
                 653  ;
                 654  ; Reset-Vektor in der Seite 3 und
                 655  ; in der LC zum Warmstart umsetzen
                 656  ;
103B: A9 50      657  LCOK     LDA  #<WRMSTART
103D: 8D FC FF   658          STA  RESET
1040: 8D F2 03   659          STA  SOFTEV
1043: A9 10      660          LDA  #>WRMSTART
1045: 8D FD FF   661          STA  RESET+1
1048: 8D F3 03   662          STA  SOFTEV+1
104B: 49 A5      663          EOR  #$A5
104D: 8D F4 03   664          STA  SOFTEV+2
                 665  ;
                 666  ; Warmstart
                 667  ;
1050: AD 82 C0   668  WRMSTART LDA  ROM        ;ROM's ein
1053: 20 89 FE   669          JSR  SETKBD
1056: 20 93 FE   670          JSR  SETVID
1059: 20 58 FC   671          JSR  HOME
105C: 20 8A 15   672          JSR  PRINT4
105F: C2 D9 D4   673     ASC  "BYTE-BLIZZARD  VERSION 1.0 (16 SEKTOREN)"
1062: C5 AD C2 CC C9 DA DA C1
106A: D2 C4 A0 A0 D6 C5 D2 D3
1072: C9 CF CE A0 B1 AE B0 A0
```

```
107A: A8 B1 B6 A0 D3 C5 CB D4
1082: CF D2 C5 CE A9
1087: A8 C3 A9  674     DCI  "(C) 1985  DR. JUERGEN KEHREL, HEIDELBERG"
108A: A0 B1 B9 B8 B5 A0 A0 C4
1092: D2 AE A0 CA D5 C5 D2 C7
109A: C5 CE A0 CB C5 C8 D2 C5
10A2: CC AC A0 C8 C5 C9 C4 C5
10AA: CC C2 C5 D2 47
10AF: 20 B2 15  675           JSR  ORIGNORM
10B2: 20 AF 16  676           JSR  SL0TDR1
10B5: 85 07     677           STA  DRIVE1
10B7: A5 08     678           LDA  SL0T2
10B9: 85 06     679           STA  SL0T1      ;herüberkopieren
10BB: 20 C5 15  680           JSR  C0PYNORM
10BE: 20 B7 16  681           JSR  SL0TDR2
10C1: 20 20 17  682           JSR  SL0TM0DI   ;Adressen modifiz.
10C4: A0 10     683  WRMSTRT1 LDY  #$10       ;HTAB
10C6: A9 0B     684           LDA  #$0B       ;Zeile 11
10C8: 20 85 15  685           JSR  PRINT2
10CB: D4 D2 C1  686           ASC  „TRACK:"
10CE: C3 CB BA
10D1: 8D        687           HEX  8D
10D2: A0 A0 B0  688     ASC  "  0000000000111111111122222222233333"
10D5: B0 B0 B0 B0 B0 B0 B0 B0
10DD: B0 B1 B1 B1 B1 B1 B1 B1
10E5: B1 B1 B1 B2 B2 B2 B2 B2
10ED: B2 B2 B2 B2 B2 B3 B3 B3
10F5: B3 B3
10F7: 8D        689           HEX  8D
10F8: A0 A0 B0  690     ASC  "  01234567890123456789012345678901234"
10FB: B1 B2 B3 B4 B5 B6 B7 B8
1103: B9 B0 B1 B2 B3 B4 B5 B6
110B: B7 B8 B9 B0 B1 B2 B3 B4
1113: B5 B6 B7 B8 B9 B0 B1 B2
111B: B3 B4
111D: 8D 8D 8D  691           HEX  8D8D8D
1120: A0 AD AD  692     ASC  " ---------------------------------------"
1123: AD AD AD AD AD AD AD AD
112B: AD AD AD AD AD AD AD AD
1133: AD AD AD AD AD AD AD AD
113B: AD AD AD AD AD AD AD AD
1143: AD AD AD
1146: 8D        693           HEX  8D
1147: A0 CC A0  694     ASC  " L = LESEN, S = SCHREIBEN, P = PRUEFEN"
114A: BD A0 CC C5 D3 C5 CE AC
1152: A0 D3 A0 BD A0 D3 C3 C8
115A: D2 C5 C9 C2 C5 CE AC A0
1162: D0 A0 BD A0 D0 D2 D5 C5
116A: C6 C5 CE
116D: 8D        695           HEX  8D
116E: A0 DE A0  696     ASC  " ↑ = IM SPEICHER,  * = FEHLER, . = GUT"
1171: BD A0 C9 CD A0 D3 D0 C5
1179: C9 C3 C8 C5 D2 AC A0 A0
1181: AA A0 BD A0 C6 C5 C8 CC
```

```
1189: C5 D2 AC A0 AE A0 BD A0
1191: C7 D5 D4
1194: 8D        697           HEX  8D
1195: A0 AD AD  698      ASC  " ------------------------------------"
1198: AD AD AD AD AD AD AD AD
11A0: AD AD AD AD AD AD AD AD
11A8: AD AD AD AD AD AD AD AD
11B0: AD AD AD AD AD AD AD AD
11B8: AD AD AD
11BB: 8D 8D     699           HEX  8D8D
11BD: A0 A0 C2  700           ASC  "  BEIDE DISKETTEN EINLEGEN"
11C0: C5 C9 C4 C5 A0 C4 C9 D3
11C8: CB C5 D4 D4 C5 CE A0 C5
11D0: C9 CE CC C5 C7 C5 CE
11D7: 8D 8D     701           HEX  8D8D
11D9: A0 A0 A8  702           DCI  "  (K)OPIEREN  (B)OOTEN  ‹ESC› ?"
11DC: CB A9 CF D0 C9 C5 D2 C5
11E4: CE A0 A0 A8 C2 A9 CF CF
11EC: D4 C5 CE A0 A0 A0 BC C5
11F4: D3 C3 BE A0 3F
11F9: 2C 10 C0  703           BIT  STROBE
11FC: 20 0C FD  704  AUSWAHL  JSR  RDKEY      ;Tastendruck
11FF: C9 CB     705           CMP  #"K"       ;Kopieren
1201: F0 2A     706           BEQ  COPY
1203: C9 C2     707           CMP  #"B"       ;Beenden
1205: F0 0C     708           BEQ  REBOOT
1207: C9 9B     709           CMP  #$9B       ;‹ESC›
1209: D0 03     710           BNE  SONICHT
120B: 4C 50 10  711           JMP  WRMSTART
                712  ;
120E: 20 DD FB  713  SONICHT  JSR  BELL       ;nicht möglich
1211: F0 E9     714           BEQ  AUSWAHL    ;immer
                715  ;
                716  ; Reboot-Routine im aktuellen Slot
                717  ;
1213: 20 58 FC  718  REBOOT   JSR  HOME
1216: AD F3 03  719           LDA  SOFTEV+1   ;Power-Up-Vektor
1219: 8D F4 03  720           STA  SOFTEV+2   ;zerstören
121C: A5 06     721           LDA  SLOT1
121E: 4A        722           LSR             ; *16
121F: 4A        723           LSR
1220: 4A        724           LSR
1221: 4A        725           LSR
1222: 09 C0     726           ORA  #$C0
1224: 85 CF     727           STA  PTR+1
1226: A9 00     728           LDA  #$00
1228: 85 CE     729           STA  PTR
122A: 6C CE 00  730           JMP  (PTR)      ;Sprung z. Controller
                731  ;
                732  ; Diskettenkopie
                733  ;
122D: A9 0E     734  COPY     LDA  #$0E       ;Zeile 14
122F: 20 5B FB  735           JSR  TABV
1232: A9 00     736           LDA  #$00       ;ab Zeilenbeginn
```

```
1234: 85 24     737           STA  CH
1236: 20 9C FC  738           JSR  CLREOL     ;Zeile löschen
1239: A9 00     739           LDA  #$00       ;initialisieren
123B: 8D 37 0F  740           STA  TRKPOS
123E: 8D 35 0F  741           STA  TRACK0     ;Track Original
1241: 8D 36 0F  742           STA  TRACKC     ;Track Copy
1244: A9 16     743           LDA  #$16       ;Sync für GAP3 auf
1246: 85 1F     744           STA  SYNCS      ;$1A (22+4) setzen
                745 ;
1248: AD 83 C0  746 ORIGINAL  LDA  LCRAMB2    ;RAM Read Bank2
124B: AD 83 C0  747           LDA  LCRAMB2    ;RAM Write Bank2
124E: A6 06     748           LDX  SLOT1      ;Slot Original
1250: A5 07     749           LDA  DRIVE1     ;Drive Original
1252: 20 C9 14  750           JSR  DRVWAHL    ;Drive anwählen
1255: 20 DF 14  751           JSR  TRK0POS    ;auf Spur 0 posit.
1258: A9 0A     752           LDA  #10        ;11 Tracks
125A: 20 95 12  753           JSR  READTRKS   ;Tracks lesen
125D: A6 08     754           LDX  SLOT2
125F: A5 09     755           LDA  DRIVE2
1261: 20 C9 14  756           JSR  DRVWAHL    ;Drive anwählen
1264: 20 DF 14  757           JSR  TRK0POS    ;auf Track 0 zurück
1267: A9 0A     758           LDA  #10        ;11 Track schreiben
1269: 20 55 13  759           JSR  WRITETRK
                760 ;
126C: A9 02     761           LDA  #02        ;2 Durchgänge
126E: 8D 94 12  762           STA  LOOPCNT
1271: A6 06     763 COPYLOOP  LDX  SLOT1      ;Slot Original
1273: A5 07     764           LDA  DRIVE1     ;Drive Original
1275: 20 C9 14  765           JSR  DRVWAHL    ;Drive anwählen
1278: A9 0B     766           LDA  #11        ;12 Tracks/Durchgang
127A: 20 95 12  767           JSR  READTRKS   ;Tracks lesen
127D: A6 08     768           LDX  SLOT2
127F: A5 09     769           LDA  DRIVE2
1281: 20 C9 14  770           JSR  DRVWAHL    ;Drive anwählen
1284: A9 0B     771           LDA  #11        ;12 Tracks schreiben
1286: 20 55 13  772           JSR  WRITETRK
1289: CE 94 12  773           DEC  LOOPCNT    ;Fertig?
128C: D0 E3     774           BNE  COPYLOOP   ;Nein
                775 ;
128E: AD 82 C0  776           LDA  ROM        ;ROM ein
1291: 4C C4 10  777           JMP  WRMSTRT1   ;NEU
                778 ;
1294: 00        779 LOOPCNT   HEX  00
                780 ;
                781 ; Im Akku übergebene Anzahl von Tracks lesen
                782 ;
1295: 8D 34 0F  783 READTRKS  STA  TRACK      ;Anzahl der Tracks
1298: BD 89 C0  784           LDA  MOTORON,X
129B: 20 6B 15  785           JSR  PAUSE      ;warten
129E: AD 35 0F  786           LDA  TRACK0     ;Pos. Original
12A1: 8D 37 0F  787           STA  TRKPOS     ;für Fehleranzeige
12A4: 0A        788           ASL             ;*2
12A5: 85 EB     789           STA  SPURIST
                790 ;
```

```
                        791  ; Lesepuffer setzen
                        792  ;
12A7: AC 34 0F          793  NXTTRK   LDY  TRACK
12AA: B9 FF 17          794          LDA  PUFTAB,Y
12AD: 8D 32 0F          795          STA  STORE      ;Hi-Byte Grundwert
12B0: 20 3E 0F          796          JSR  MERK       ;(PTR) setzen
12B3: A9 0C             797          LDA  #$0C       ;L inv
12B5: AC 35 0F          798          LDY  TRACK0
12B8: 99 2A 07          799          STA  ZEILE14,Y
12BB: A9 50             800          LDA  #$50       ;5-Mal
12BD: 8D 3C 0F          801          STA  RDVER      ;Zähler
                        802  ;
                        803  ; Sektormerkbereich
                        804  ; FF = Neu , 00 = gelesen
                        805  ;
12C0: A9 FF             806          LDA  #$FF       ;negativ
12C2: A0 10             807          LDY  #$10       ;16 Sektoren
12C4: 88                808  SIML00P  DEY          ;Auf negativ setzen
12C5: 99 ED 0D          809          STA  SIM,Y      ;Merkbereich
12C8: D0 FA             810          BNE  SIML00P    ;Schleife
                        811  ;
                        812  ; Sektoren lesen
                        813  ;
12CA: 84 1E             814  RDSEKTOR STY  SEKTOR    ;zu lesender Sektor
12CC: A6 06             815  RDS1     LDX  SLOT1     ;Slot Original
12CE: 20 D4 0E          816  READADR  JSR  RDADR16   ;Adressfeld lesen
12D1: 90 1C             817          BCC  ADR0K     ;ohne Fehler
12D3: CE 3C 0F          818  TRY      DEC  RDVER     ;Zahl der Versuche -1
12D6: D0 F6             819          BNE  READADR   ;neu versuchen
                        820  ;
12D8: AD 82 C0          821  FEHLADR  LDA  ROM       ;ROM ein
12DB: 20 AF 15          822          JSR  ORIGINV
12DE: 4C 4F 16          823          JMP  ADRFEHL   ;Fehlermeldung ausgeben
                        824  ;
12E1: CE 3C 0F          825  TRY1     DEC  RDVER
12E4: D0 E6             826          BNE  RDS1
12E6: AD 82 C0          827          LDA  ROM
12E9: 20 AF 15          828          JSR  ORIGINV
12EC: 4C 5C 16          829          JMP  DATFEHL   ;Datenfehler
                        830  ;
                        831  ;
12EF: AD 3A 0F          832  ADR0K    LDA  RTRK      ;gelesener Track
12F2: CD 35 0F          833          CMP  TRACK0    ;erwarteter Track
12F5: D0 E1             834          BNE  FEHLADR
12F7: AC 39 0F          835          LDY  RSEC
12FA: C0 10             836          CPY  #$10      ;max. $F
12FC: B0 DA             837          BGE  FEHLADR
12FE: B9 ED 0D          838          LDA  SIM,Y
1301: F0 D0             839          BEQ  TRY       ;hatten wir schon
1303: 98                840          TYA
1304: 0D 32 0F          841          ORA  STORE     ;Ablageort bestimmen
1307: 8D 33 0F          842          STA  ZIELHI
                        843  ;
                        844  ; Daten in den Pufferbereich einlesen
                        845  ;
```

```
130A: 20 FD 0D  846           JSR  READ16      ;Datenfeld lesen
130D: B0 D2     847           BCS  TRY1        ;Fehler, nächster Versuch
130F: AC 39 0F  848           LDY  RSEC        ;Sektor markieren
1312: A9 00     849           LDA  #$00        ;auf 00 (positiv) setzen
1314: 99 ED 0D  850           STA  SIM,Y
1317: A4 1E     851           LDY  SEKTOR      ;nächsten Sektor
1319: C8        852           INY
131A: C0 10     853           CPY  #$10        ;schon 16 gelesen?
131C: D0 AC     854           BNE  RDSEKTOR    ;Nein, also weiter
                855 ;
                856 ; nächsten Track lesen
                857 ;
131E: AC 35 0F  858           LDY  TRACK0
1321: A9 DE     859           LDA  #"↑"
1323: 99 2A 07  860           STA  ZEILE14,Y
1326: A6 06     861           LDX  SLOT1
1328: C8        862           INY              ;nächste Spur
1329: 8C 35 0F  863           STY  TRACK0
132C: 8C 37 0F  864           STY  TRKPOS
132F: 98        865           TYA
1330: 0A        866           ASL              ;*2
1331: 20 EE 14  867           JSR  SEEKABS
1334: CE 34 0F  868           DEC  TRACK
1337: 30 0A     869           BMI  ALLTRK
1339: AD 00 C0  870           LDA  KEY
133C: C9 9B     871           CMP  #$9B        ;‹ESC›
133E: F0 07     872           BEQ  ESCENDE
1340: 4C A7 12  873           JMP  NXTTRK      ;nächsten Track lesen
                874 ;
1343: BD 88 C0  875 ALLTRK    LDA  MOTOROFF,X  ;Motor aus
1346: 60        876           RTS
                877 ;
                878 ; vorzeitiger Abbruch
                879 ;
1347: 2C 10 C0  880 ESCENDE   BIT  STROBE
134A: 20 D4 14  881           JSR  MOTOFF      ;Motoren aus
134D: AD 82 C0  882           LDA  ROM
1350: 68        883           PLA
1351: 68        884           PLA              ;1* JSR
1352: 4C C4 10  885           JMP  WRMSTRT1
                886 ;
                887 ;
                888 ; Kopie schreiben/verifizieren
                889 ;
                890 ;
1355: 8D 34 0F  891 WRITETRK  STA  TRACK       ;Anzahl Tracks
1358: BD 89 C0  892           LDA  MOTORON,X
135B: 20 6B 15  893           JSR  PAUSE
135E: AD 36 0F  894           LDA  TRACKC
1361: 8D 37 0F  895           STA  TRKPOS
1364: 0A        896           ASL              ;*2
1365: 85 EB     897           STA  SPURIST
                898 ;
                899 ; Schreibpuffer setzen
                900 ;
```

```
1367: AC 34 0F  901 WRNEXT  LDY  TRACK
136A: B9 FF 17  902         LDA  PUFTAB,Y
136D: 8D 32 0F  903         STA  STORE      ;Hi-Byte Grundwert
1370: 20 3E 0F  904         JSR  MERK       ;(PTR) setzen
1373: A9 03     905         LDA  #$03       ;3 Versuche
1375: 8D 3D 0F  906         STA  WRVERS
                907 ;
1378: A9 13     908 WRITE   LDA  #$13       ;Schreiben
137A: AC 36 0F  909         LDY  TRACKC
137D: 99 2A 07  910         STA  ZEILE14,Y
1380: A9 05     911         LDA  #$5        ;wenig Versuche
1382: 8D 3C 0F  912         STA  RDVER
                913 ;
                914 ; 1 Track formatieren und die
                915 ; Daten des Puffers schreiben
                916 ;
1385: 20 03 0C  917         JSR  WRTRK
                918 ;
                919 ;
                920 ; Track verifizieren
                921 ;
1388: A9 10     922         LDA  #$10       ;Prüfen
138A: AC 36 0F  923         LDY  TRACKC
138D: 99 2A 07  924         STA  ZEILE14,Y  ;ausgeben
                925 ;
                926 ; Formatierung der Disk überprüfen. Nächster
                927 ; lesbarer Sektor muß wieder der erste sein,
                928 ; sonst sind die Sync-Felder zu lang und der
                929 ; Track überschreibt sich selber
                930 ;
1390: A4 1F     931         LDY  SYNCS
1392: 20 EC 0D  932 VFW     JSR  WAITRTS
1395: 20 EC 0D  933         JSR  WAITRTS    ;Pause für GAP1
1398: 20 EC 0D  934         JSR  WAITRTS
139B: 48        935         PHA
139C: 68        936         PLA
139D: EA        937         NOP
139E: 88        938         DEY
139F: D0 F1     939         BNE  VFW
                940 ;
13A1: A0 0F     941         LDY  #$0F       ;16 Sektoren
13A3: 84 1E     942         STY  SEKTOR
13A5: A9 FF     943         LDA  #$FF       ;auf <>0 initial.
13A7: 99 ED 0D  944 LOESCH  STA  SIM,Y      ;Zwischenspeicher für
13AA: 88        945         DEY             ;Sektornummern
13AB: 10 FA     946         BPL  LOESCH
                947 ;
13AD: A6 08     948         LDX  SLOT2
13AF: 20 D4 0E  949         JSR  RDADR16    ;nächstes Adressfeld
13B2: B0 31     950         BCS  NEUFORM    ;nicht lesbar
                951 ;
                952 ; war es der erste Sektor?
                953 ;
13B4: AD 39 0F  954         LDA  RSEC       ;gelesener Sektor
```

```
13B7: F0 18     955           BEQ  VTRK        ;Sektor 0
13B9: 20 BC 14  956           JSR  SYNCZAHL    ;Syncs verkürzen
13BC: B0 27     957           BCS  NEUFORM     ;neu kopieren
                958 ;
13BE: AD 82 C0  959 NOTFOUND  LDA  ROM
13C1: 4C 38 16  960           JMP  FORMFEHL    ;Formatierung
                961 ;
                962 ;
                963 ; Ganzen Track lesen, alle Sektoren auf
                964 ; Vollständigkeit und Lesbarkeit prüfen
                965 ;
                966 ;
13C4: 20 D4 0E  967 RDSEKT    JSR  RDADR16
13C7: B0 0D     968           BCS  VVFE        ;Verify-Fehler
13C9: AC 39 0F  969           LDY  RSEC        ;gefundener Sektor
13CC: B9 ED 0D  970           LDA  SIM,Y       ;schon gehabt?
13CF: F0 05     971           BEQ  VVFE        ;Ja
                972 ;
13D1: 20 59 14  973 VTRK      JSR  VREAD16     ;Daten prüfen
13D4: 90 2D     974           BCC  VOKAY
                975 ;
13D6: CE 3C 0F  976 VVFE      DEC  RDVER
13D9: D0 E9     977           BNE  RDSEKT      ;neuer Versuch
                978 ;
                979 ; Verify-Fehler
                980 ;
13DB: CE 3D 0F  981           DEC  WRVERS
13DE: F0 DE     982           BEQ  NOTFOUND
13E0: A9 20     983           LDA  #$20
13E2: 8D 3C 0F  984           STA  RDVER
                985 ;
                986 ; Das Ende der Spur wird abgewartet,
                987 ; dann wird neu formatiert
                988 ;
13E5: AD 36 0F  989 NEUFORM   LDA  TRACKC
13E8: F0 0F     990           BEQ  TOWRITE
13EA: 20 D4 0E  991 NFORM1    JSR  RDADR16
13ED: B0 0D     992           BCS  NEUF1
13EF: AD 39 0F  993           LDA  RSEC
13F2: C9 0F     994           CMP  #$0F        ;letzter Sektor?
13F4: D0 06     995           BNE  NEUF1
13F6: 20 59 14  996           JSR  VREAD16     ;Daten lesen
13F9: 4C 78 13  997 TOWRITE   JMP  WRITE       ;Neu schreiben
                998 ;
13FC: CE 3C 0F  999 NEUF1     DEC  RDVER
13FF: D0 E9     1000          BNE  NFORM1
1401: F0 BB     1001          BEQ  NOTFOUND
                1002 ;
1403: AC 39 0F  1003 VOKAY    LDY  RSEC        ;gelesener Sektor
1406: A9 00     1004          LDA  #$00        ;markieren
1408: 99 ED 0D  1005          STA  SIM,Y
                1006 ;
140B: C6 1E     1007          DEC  SEKTOR      ;nächsten Sektor
140D: 10 B5     1008          BPL  RDSEKT      ;weiterprüfen
                1009 ;
```

```
                    1010 ; Zurück zu Sektor 0, damit die
                    1011 ; Sektoren benachbarter Tracks
                    1012 ; in der richtigen Anordnung
                    1013 ; nebeneinander liegen
                    1014 ;
140F: A9 20         1015          LDA  #$20
1411: 8D 3C 0F      1016          STA  RDVER
1414: CE 3C 0F      1017 SEEK0    DEC  RDVER
1417: F0 A5         1018          BEQ  NOTFOUND
1419: 20 D4 0E      1019          JSR  RDADR16
141C: B0 F6         1020          BCS  SEEK0
141E: AD 39 0F      1021          LDA  RSEC
1421: D0 F1         1022          BNE  SEEK0      ;nicht Sektor 0
1423: 20 59 14      1023          JSR  VREAD16    ;Daten lesen
1426: B0 EC         1024          BCS  SEEK0
                    1025 ;
                    1026 ; Verify erfolgreich, also anzeigen
                    1027 ;
1428: A9 AE         1028          LDA  #$AE       ;„."
142A: AC 36 0F      1029          LDY  TRACKC
142D: 99 2A 07      1030          STA  ZEILE14,Y  ;ausgeben
1430: D0 03         1031          BNE  ONTRK0     ;Track 0?
1432: 20 BC 14      1032          JSR  SYNCZAHL   ;Synczahl verringern
1435: A6 08         1033 ONTRK0   LDX  SLOT2
1437: C8            1034          INY             ;nächste Spur
1438: 8C 36 0F      1035          STY  TRACKC
143B: 8C 37 0F      1036          STY  TRKPOS
143E: 98            1037          TYA
143F: 0A            1038          ASL             ;*2
1440: 20 EE 14      1039          JSR  SEEKABS
1443: CE 34 0F      1040          DEC  TRACK
1446: 30 0D         1041          BMI  ALLWRT
1448: AD 00 C0      1042          LDA  KEY
144B: C9 9B         1043          CMP  #$9B       ;‹ESC›?
144D: D0 03         1044          BNE  GOWRNXT    ;Nein, weiter
144F: 4C 47 13      1045          JMP  ESCENDE    ;Abbruch
                    1046 ;
1452: 4C 67 13      1047 GOWRNXT  JMP  WRNEXT
                    1048 ;
1455: BD 88 C0      1049 ALLWRT   LDA  MOTOROFF,X ;Motor aus
1458: 60            1050          RTS
                    1051 ;
                    1052 ;
                    1053 ; alle Datenbytes lesen und Checksum bilden
                    1054 ; gelesenen Datenbytes werden verworfen
                    1055 ;
1459: A0 20         1056 VREAD16  LDY  #$20       ;Versuchszahl
145B: 88            1057 VDATA    DEY
145C: F0 5C         1058          BEQ  VDERR      ;Lesefehler
                    1059 ;
145E: BD 8C C0      1060 VDT      LDA  DATASTRB,X ;Datenprolog lesen
1461: 10 FB         1061          BPL  VDT        ;warten
1463: C9 D5         1062 VDT0     CMP  #$D5
1465: D0 F4         1063          BNE  VDATA      ;noch nicht
```

```
1467: EA        1064          NOP             ;Pause
1468: BD 8C C0  1065 VDT1     LDA  DATASTRB,X ;lesen
146B: 10 FB     1066          BPL  VDT1       ;warten
146D: C9 AA     1067          CMP  #$AA
146F: D0 F2     1068          BNE  VDT0       ;muß kommen
1471: A0 56     1069          LDY  #$56       ;Datenbytes
1473: BD 8C C0  1070 VDT2     LDA  DATASTRB,X ;lesen
1476: 10 FB     1071          BPL  VDT2       ;warten
1478: C9 AD     1072          CMP  #$AD
147A: D0 E7     1073          BNE  VDT0       ;muß kommen
                1074 ;
                1075 ;
                1076 ; Datenfeld lesen für Checksum-Prüfung
                1077 ; Daten werden nicht gespeichert
                1078 ;
147C: 84 18     1079          STY  INDEX
147E: A9 00     1080          LDA  #$00
1480: BC 8C C0  1081 VDAT1    LDY  DATASTRB,X
1483: 10 FB     1082          BPL  VDAT1
1485: 59 00 0F  1083          EOR  LUET-$96,Y
1488: EA        1084          NOP
1489: C6 18     1085          DEC  INDEX
148B: D0 F3     1086          BNE  VDAT1
                1087 ;
148D: BC 8C C0  1088 VDAT2    LDY  DATASTRB,X
1490: 10 FB     1089          BPL  VDAT2
1492: 59 00 0F  1090          EOR  LUET-$96,Y
1495: EA        1091          NOP
1496: E6 18     1092          INC  INDEX
1498: D0 F3     1093          BNE  VDAT2
                1094 ;
                1095 ; Jetzt Checksum
                1096 ;
149A: BC 8C C0  1097 VDAT3    LDY  DATASTRB,X
149D: 10 FB     1098          BPL  VDAT3
149F: D9 00 0F  1099          CMP  LUET-$96,Y
14A2: D0 16     1100          BNE  VDERR
                1101 ;
                1102 ; Datenfeld-Epilog prüfen
                1103 ;
14A4: EA        1104          NOP
14A5: BD 8C C0  1105 VDAT4    LDA  DATASTRB,X
14A8: 10 FB     1106          BPL  VDAT4
14AA: C9 DE     1107          CMP  #$DE
14AC: D0 0C     1108          BNE  VDERR
14AE: EA        1109          NOP
14AF: BD 8C C0  1110 VDAT5    LDA  DATASTRB,X
14B2: 10 FB     1111          BPL  VDAT5
14B4: C9 AA     1112          CMP  #$AA
14B6: D0 02     1113          BNE  VDERR
14B8: 18        1114          CLC
14B9: 60        1115          RTS
                1116 ;
14BA: 38        1117 VDERR    SEC
```

```
14BB: 60          1118          RTS
                  1119 ;
                  1120 ; Zahl der Sync-Bytes verringern
                  1121 ;
14BC: A9 0D       1122 SYNCZAHL LDA  #$0D        ;noch 14+4 Syncs?
14BE: C5 1F       1123          CMP  SYNCS       ;akt. Anzahl
14C0: A5 1F       1124          LDA  SYNCS
14C2: E9 01       1125          SBC  #$01        ;-2 bei CC, -1 bei CS
14C4: 85 1F       1126          STA  SYNCS       ;abspeichern
14C6: C9 06       1127          CMP  #$06        ;noch 10 (6+4) ?
14C8: 60          1128          RTS              ;CS = OK, CC = Fehler
                  1129 ;
                  1130 ; gewünschtes Laufwerk anwählen
                  1131 ;
14C9: 4A          1132 DRVWAHL  LSR              ;gerade/ungerade?
14CA: 90 04       1133          BCC  DRV2        ;gerade -> Drive 2
14CC: BD 8A C0    1134          LDA  DRV1ENAB,X  ;Drive 1 an
14CF: 60          1135          RTS
                  1136 ;
14D0: BD 8B C0    1137 DRV2     LDA  DRV2ENAB,X  ;Drive 2 an
14D3: 60          1138          RTS
                  1139 ;
                  1140 ; Motor für Orig. und Kopie aus
                  1141 ;
14D4: A6 06       1142 MOTOFF   LDX  SLOT1
14D6: BD 88 C0    1143          LDA  MOTOROFF,X
14D9: A6 08       1144          LDX  SLOT2
14DB: BD 88 C0    1145          LDA  MOTOROFF,X
14DE: 60          1146          RTS
                  1147 ;
                  1148 ; Auf Spur 0 positionieren.
                  1149 ;
14DF: BD 8E C0    1150 TRKOPOS  LDA  INPUTMOD,X
14E2: BD 8C C0    1151          LDA  DATASTRB,X  ;Schleuse normalisieren
14E5: BD 89 C0    1152          LDA  MOTORON,X
14E8: A9 50       1153          LDA  #$50        ;Spurist * 2 auf 80 (40)
14EA: 85 EB       1154          STA  SPURIST
14EC: A9 00       1155          LDA  #$00        ;Spursoll * 2 auf 00
                  1156 ;
                  1157 ; positioniert auf die gewünschte Spur
                  1158 ;
14EE: 86 17       1159 SEEKABS  STX  TEMP        ;Slot*16
14F0: 85 EC       1160          STA  SPURSOLL
14F2: C5 EB       1161          CMP  SPURIST
14F4: F0 59       1162          BEQ  SPURDA      ;gleich, Prozedur erfüllt
14F6: A9 00       1163          LDA  #$00        ;Delay-Index auf Maximum
14F8: 85 CE       1164          STA  PTR         ;Durchlaufzähler
14FA: A0 03       1165          LDY  #$03        ;alle Phasen aus
14FC: 18          1166          CLC
14FD: 98          1167 PHASOUT  TYA
14FE: 20 44 15    1168          JSR  ARMMOV1
1501: 88          1169          DEY
1502: 10 F9       1170          BPL  PHASOUT
                  1171 ;
```

```
1504: A5 EB      1172 POSIT    LDA  SPURIST    ;Track jetzt *2
1506: 85 CF      1173          STA  PTR+1      ;retten
1508: 38         1174          SEC
1509: E5 EC      1175          SBC  SPURSOLL   ;- gewünschte Spur
150B: F0 31      1176          BEQ  ISTDA      ;gleich, also Ende
150D: B0 06      1177          BCS  AUSSEN     ;> Arm zu weit innen
                 1178 ;
                 1179 ; nach innen bewegen
                 1180 ;
150F: 49 FF      1181          EOR  #$FF       ;Vorzeichenwechsel, -1
1511: E6 EB      1182          INC  SPURIST    ;akt. Spur erhöht
1513: 90 04      1183          BCC  DELAYNDX   ;immer
                 1184 ;
                 1185 ; nach außen bewegen
                 1186 ;
1515: 69 FE      1187 AUSSEN   ADC  #$FE       ;nach außen. -1,
1517: C6 EB      1188          DEC  SPURIST    ; akt. Spur +1
1519: C5 CE      1189 DELAYNDX CMP  PTR        ;Vergl Diff-1 / Delay
151B: 90 02      1190          BCC  DIFFKLN    ;Differenz kleiner
151D: A5 CE      1191          LDA  PTR        ;Delay-Index kleiner
151F: C9 08      1192 DIFFKLN  CMP  #$08       ;< $08 (Maximalwert)
1521: B0 01      1193          BCS  DELMAX     ;Nein, Y unverändert
1523: A8         1194          TAY             ;Akku neuer Delay-Index
1524: 38         1195 DELMAX   SEC             ;Phase an zum Track
1525: 20 42 15   1196          JSR  ARMMOVER   ;Arm bewegen
1528: B9 50 15   1197          LDA  PHONDEL,Y  ;Phase ON Delay
152B: 20 60 15   1198          JSR  MSWAIT     ;warten
152E: A5 CF      1199          LDA  PTR+1      ;alte Spurnummer
1530: 18         1200          CLC             ;diese Phase aus
1531: 20 44 15   1201          JSR  ARMMOV1    ;ausschalten
1534: B9 58 15   1202          LDA  PHOFFDEL,Y ;Phase OFF Delay
1537: 20 60 15   1203          JSR  MSWAIT     ;warten
153A: E6 CE      1204          INC  PTR        ;Durchlaufzähler +1
153C: D0 C6      1205          BNE  POSIT      ;immer, neuer Durchgang
                 1206 ;
153E: 20 60 15   1207 ISTDA    JSR  MSWAIT     ;noch warten
1541: 18         1208          CLC             ;dann diese Phase aus
                 1209 ;
                 1210 ; Phasen-Adressierung.
                 1211 ;
1542: A5 EB      1212 ARMMOVER LDA  SPURIST    ;Einsprung „Track jetzt"
1544: 29 03      1213 ARMMOV1  AND  #$03       ;Phasennr. ausblenden
1546: 2A         1214          ROL             ;C=1 ON, C=0 OFF
1547: 05 17      1215          ORA  TEMP       ;Slot*16 einblenden
1549: AA         1216          TAX             ;Index Phasenansteuerung
154A: BD 80 C0   1217          LDA  PHASE00F,X ;Phase schalten
154D: A6 17      1218          LDX  TEMP       ;Slot*16 zurück
154F: 60         1219 SPURDA   RTS
                 1220 ;
                 1221 ; Verzögerungstabellen für Armbewegungen
                 1222 ;
1550: 01 30 28   1223 PHONDEL  HEX  01302824201E1D1C ;Verzögerung Ein
1553: 24 20 1E 1D 1C
1558: 70 2C 26   1224 PHOFFDEL HEX  702C26221F1E1D1C ;Verzögerung Aus
155B: 22 1F 1E 1D 1C
                 1225 ;
```

```
                    1226 ; Pause für Kopfbewegungen
                    1227 ;
1560: A2 12         1228 MSWAIT   LDX  #$12        ;kleine Schleife
1562: CA            1229 MSW1     DEX
1563: D0 FD         1230          BNE  MSW1
1565: 38            1231          SEC
1566: E9 01         1232          SBC  #$01        ;große Schleife
1568: D0 F6         1233          BNE  MSWAIT
156A: 60            1234          RTS
                    1235 ;
                    1236 ; Pause für Motoranlauf
                    1237 ;
156B: A0 05         1238 PAUSE    LDY  #$05        ;ca. 0,83 sec
156D: A9 FF         1239 PAUSE1   LDA  #$FF
156F: 38            1240          SEC
1570: 48            1241 PAUSE2   PHA
1571: E9 01         1242 PAUSE3   SBC  #$01        ;wie Monitor
1573: D0 FC         1243          BNE  PAUSE3
1575: 68            1244          PLA
1576: E9 01         1245          SBC  #$01
1578: D0 F6         1246          BNE  PAUSE2
157A: 88            1247          DEY
157B: D0 F0         1248          BNE  PAUSE1
157D: 60            1249          RTS
                    1250 ;
                    1251 ; Druckroutine, mehrere Einsprünge
                    1252 ;
157E: 20 80 FE      1253 PRINT    JSR  SETINV
1581: A9 15         1254          LDA  #$15
1583: A0 02         1255 PRINT1   LDY  #$02
1585: 20 5B FB      1256 PRINT2   JSR  TABV
1588: 84 24         1257 PRINT3   STY  CH
158A: 68            1258 PRINT4   PLA
158B: 85 CE         1259          STA  PTR
158D: 68            1260          PLA
158E: 85 CF         1261          STA  PTR+1
1590: A0 00         1262          LDY  #$00
1592: E6 CE         1263 PR1      INC  PTR
1594: D0 02         1264          BNE  PR2
1596: E6 CF         1265          INC  PTR+1
1598: B1 CE         1266 PR2      LDA  (PTR),Y
159A: 48            1267          PHA
159B: 09 80         1268          ORA  #$80        ;Bit 7 setzen
159D: 20 F0 FD      1269          JSR  COUT1       ;ausgeben
15A0: 68            1270          PLA
15A1: 30 EF         1271          BMI  PR1         ;Positiv Endekennung
15A3: A5 CF         1272          LDA  PTR+1       ;Rücksprungadresse
15A5: 48            1273          PHA              ;auf den Stack
15A6: A5 CE         1274          LDA  PTR         ;schieben und mit
15A8: 48            1275          PHA
15A9: 60            1276          RTS              ;RTS anspringen
                    1277 ;
                    1278 ; Original- und Copyanzeige löschen
                    1279 ;
```

```
15AA: 20 B2 15   1280 NORMAL   JSR  ORIGNORM
15AD: D0 16      1281          BNE  COPYNORM   ;immer
                 1282 ;
                 1283 ; Original einlegen
                 1284 ;
15AF: 20 80 FE   1285 ORIGINV  JSR  SETINV
15B2: A9 04      1286 ORIGNORM LDA  #$04
15B4: 20 83 15   1287          JSR  PRINT1
15B7: CF D2 C9   1288          DCI  "ORIGINAL"
15BA: C7 C9 CE C1 4C
15BF: 4C 84 FE   1289          JMP  SETNORM
                 1290 ;
                 1291 ; Copy einlegen
                 1292 ;
15C2: 20 80 FE   1293 COPYINV  JSR  SETINV
15C5: A9 07      1294 COPYNORM LDA  #$07
15C7: A0 05      1295          LDY  #$05
15C9: 20 85 15   1296          JSR  PRINT2
15CC: CB CF D0   1297          DCI  "KOPIE"
15CF: C9 45
15D1: 4C 84 FE   1298          JMP  SETNORM
                 1299 ;
                 1300 ; Fehlermeldungen
                 1301 ;
                 1302 ; Programm benötigt 64K
                 1303 ;
15D4: 20 58 FC   1304 LCERROR  JSR  HOME
15D7: 20 8A 15   1305          JSR  PRINT4
15DA: D0 D2 CF   1306          ASC  "PROGRAMM BENOETIGT 16K-KARTE !"
15DD: C7 D2 C1 CD CD A0 C2 C5
15E5: CE CF C5 D4 C9 C7 D4 A0
15ED: B1 B6 CB AD CB C1 D2 D4
15F5: C5 A0 A1
15F8: 87 87 87   1307          HEX  8787878D8D
15FB: 8D 8D
15FD: BE BE D4   1308          DCI  "››TASTENDRUCK BOOTET‹‹"
1600: C1 D3 D4 C5 CE C4 D2 D5
1608: C3 CB A0 C2 CF CF D4 C5
1610: D4 BC 3C
1613: 20 0C FD   1309          JSR  RDKEY
1616: 4C 13 12   1310          JMP  REBOOT
                 1311 ;
                 1312 ; Disk schreibgeschützt
                 1313 ;
1619: BD 88 C0   1314 WPROT    LDA  MOTOROFF,X ;Motor aus
161C: 68         1315          PLA
161D: 68         1316          PLA
161E: 8D 82 C0   1317          STA  ROM
1621: 20 C2 15   1318          JSR  COPYINV
1624: 20 7E 15   1319          JSR  PRINT
1627: D3 C3 C8   1320          DCI  "SCHREIBSCHUTZ-"
162A: D2 C5 C9 C2 D3 C3 C8 D5
1632: D4 DA 2D
1635: 4C 65 16   1321          JMP  ERROR
                 1322 ;
```

```
                    1323 ; Formatierung nicht möglich
                    1324 ;
1638: 20 C2 15      1325 FORMFEHL JSR  COPYINV
163B: 20 7E 15      1326          JSR  PRINT
163E: C6 CF D2      1327          DCI  "FORMATIERUNGS-"
1641: CD C1 D4 C9 C5 D2 D5 CE
1649: C7 D3 2D
164C: 4C 65 16      1328          JMP  ERROR
                    1329 ;
                    1330 ; Adressfeld-Fehler
                    1331 ;
164F: 20 7E 15      1332 ADRFEHL  JSR  PRINT
1652: C1 C4 D2      1333          DCI  "ADRESS-"
1655: C5 D3 D3 2D
1659: 4C 65 16      1334          JMP  ERROR
                    1335 ;
                    1336 ; Datenfeld fehlerhaft
                    1337 ;
165C: 20 7E 15      1338 DATFEHL  JSR  PRINT
165F: C4 C1 D4      1339          DCI  "DATEN-"
1662: C5 CE 2D
                    1340 ;
                    1341 ; Fehlerposition ausgeben
                    1342 ;
1665: 68            1343 ERROR    PLA
1666: 68            1344          PLA
1667: AE 37 0F      1345          LDX  TRKPOS
166A: A9 AA         1346          LDA  #"*"        :Fehler
166C: 9D 2A 07      1347          STA  ZEILE14,X
166F: 20 8A 15      1348          JSR  PRINT4
1672: 87            1349          HEX  87          ;Bell
1673: C6 C5 C8      1350          DCI  "FEHLER"
1676: CC C5 52
1679: 20 84 FE      1351          JSR  SETNORM
167C: 20 9C FC      1352          JSR  CLREOL
                    1353 ;
                    1354 ; Abbruch, aber Slot und Drive merken
                    1355 ;
167F: 20 D4 14      1356          JSR  MOTOFF
1682: A9 17         1357          LDA  #$17        ;Zeile 16
1684: 20 83 15      1358          JSR  PRINT1
1687: C2 C9 D4      1359          DCI  "BITTE EINE TASTE DRUECKEN"
168A: D4 C5 A0 C5 C9 CE C5 A0
1692: D4 C1 D3 D4 C5 A0 C4 D2
169A: D5 C5 C3 CB C5 4E
16A0: 20 9C FC      1360          JSR  CLREOL
16A3: 2C 10 C0      1361          BIT  STROBE
16A6: 20 0C FD      1362          JSR  RDKEY
16A9: 20 AA 15      1363          JSR  NORMAL
16AC: 4C C4 10      1364          JMP  WRMSTRT1
                    1365 ;
                    1366 ; Slot und Drive festlegen
                    1367 ;
16AF: A9 00         1368 SLOTDR1  LDA  #$00        ;initialisieren
```

```
16B1: 85 08     1369          STA  SLOT2
16B3: 85 07     1370          STA  DRIVE1
16B5: 85 17     1371          STA  TEMP
16B7: 20 8A 15  1372 SLOTDR2  JSR  PRINT4
16BA: A0 D3 D4  1373          DCI  " STECKPLATZ:"
16BD: C5 C3 CB D0 CC C1 D4 DA
16C5: 3A
16C6: A0 B8     1374          LDY  #$B8       ;„8" Obergrenze
16C8: 20 F6 16  1375          JSR  ANSWER
16CB: 0A        1376          ASL             ;; 16
16CC: 0A        1377          ASL
16CD: 0A        1378          ASL
16CE: 0A        1379          ASL
16CF: C5 08     1380          CMP  SLOT2      ;am selben Controller
16D1: 85 08     1381          STA  SLOT2      ;nur mit 2 Drives
16D3: D0 04     1382          BNE  DRIVE      ;an verschiedenen Contr.
16D5: A5 07     1383          LDA  DRIVE1
16D7: 85 17     1384          STA  TEMP
16D9: A0 0D     1385 DRIVE    LDY  #$0D       ;HTAB
16DB: 20 88 15  1386          JSR  PRINT3
16DE: CC C1 D5  1387          DCI  "LAUFWERK:"
16E1: C6 D7 C5 D2 CB 3A
16E7: A0 B3     1388          LDY  #$B3       ;„3" Obergrenze
16E9: 20 F6 16  1389          JSR  ANSWER
16EC: 85 09     1390          STA  DRIVE2
16EE: 60        1391          RTS
                1392 ;
                1393 ; Abbruch durch ‹ESC›
                1394 ;
16EF: 68        1395 ABBRUCH  PLA             ;2* JSR vom Stack
16F0: 68        1396          PLA
16F1: 68        1397          PLA
16F2: 68        1398          PLA
16F3: 4C 50 10  1399          JMP  WRMSTART
                1400 ;
                1401 ; Antwort von der Tastatur
                1402 ; Obergrenze (‹) im Y-Register
                1403 ;
16F6: 2C 10 C0  1404 ANSWER   BIT  STROBE     ;Tastatur vorbereiten
16F9: 84 CE     1405          STY  PTR        ;Obergrenze merken
16FB: 20 0C FD  1406 TASTE    JSR  RDKEY      ;Tastendruck
16FE: C9 9B     1407          CMP  #$9B       ;‹ESC›?
1700: F0 ED     1408          BEQ  ABBRUCH    ;Ja
1702: C9 B1     1409          CMP  #$B1       ;„1"
1704: 90 0A     1410          BLT  FEHL       ;zu klein
1706: C5 CE     1411          CMP  PTR        ;Obergrenze
1708: B0 06     1412          BGE  FEHL       ;zu groß
170A: 29 0F     1413          AND  #$0F       ;%00001111
170C: C5 17     1414          CMP  TEMP       ;1 Controller: 2 Drives
170E: D0 05     1415          BNE  OK
                1416 ;
1710: 20 DD FB  1417 FEHL     JSR  BELL       ;Piep
1713: F0 E6     1418          BEQ  TASTE      ;neuer Versuch
                1419 ;
```

```
                    1420 ; Tastendruck ausgeben
                    1421 ;
1715: 48            1422 OK       PHA             ;retten
1716: 09 B0         1423          ORA  #$B0       ;-> ASCII
1718: 20 F0 FD      1424          JSR  COUT1      ;ausgeben
171B: 20 8E FD      1425          JSR  CROUT      ;<cr>
171E: 68            1426          PLA             ;zurück
171F: 60            1427          RTS             ;das war's
                    1428 ;
                    1429 ;
                    1430 ; Code-Modifikation für Slot
                    1431 ;
1720: A5 06         1432 SLOTMODI LDA  SLOT1
1722: 18            1433          CLC
1723: 69 80         1434          ADC  #$80
1725: 85 17         1435          STA  TEMP
1727: A2 00         1436          LDX  #$00
1729: A0 01         1437          LDY  #$01
172B: BD 65 17      1438 OSLOTM   LDA  OSLM0,X    ;Original
172E: 85 CE         1439          STA  PTR
1730: BD 66 17      1440          LDA  OSLM0+1,X
1733: F0 0E         1441          BEQ  CSLOTMOD
1735: 85 CF         1442          STA  PTR+1
1737: B1 CE         1443          LDA  (PTR),Y
1739: 29 0F         1444          AND  #$0F
173B: 05 17         1445          ORA  TEMP
173D: 91 CE         1446          STA  (PTR),Y
173F: E8            1447          INX
1740: E8            1448          INX
1741: D0 E8         1449          BNE  OSLOTM
                    1450 ;
1743: A5 08         1451 CSLOTMOD LDA  SLOT2
1745: 18            1452          CLC
1746: 69 80         1453          ADC  #$80
1748: 85 17         1454          STA  TEMP
174A: A2 00         1455          LDX  #$00
174C: BD 79 17      1456 CSLOTM   LDA  CSLM0,X    ;Copy
174F: 85 CE         1457          STA  PTR
1751: BD 7A 17      1458          LDA  CSLM0+1,X
1754: F0 0E         1459          BEQ  MODEND
1756: 85 CF         1460          STA  PTR+1
1758: B1 CE         1461          LDA  (PTR),Y
175A: 29 0F         1462          AND  #$0F
175C: 05 17         1463          ORA  TEMP
175E: 91 CE         1464          STA  (PTR),Y
1760: E8            1465          INX
1761: E8            1466          INX
1762: D0 E8         1467          BNE  CSLOTM
                    1468 ;
1764: 60            1469 MODEND   RTS
                    1470 ;
                    1471 ; Tabelle der zu modif. Adressen
                    1472 ;
                    1473 ;
```

```
1765: 38 0E     1474 OSLM0    ADR  RDNIBBL
1767: 47 0E     1475          ADR  RDNIBBL1
1769: 5B 0E     1476          ADR  RDNIBBL2
176B: 6F 0E     1477          ADR  RDNIBBL3
176D: 83 0E     1478          ADR  RDNIBBL4
176F: 96 0E     1479          ADR  RDNIBBL5
1771: A8 0E     1480          ADR  RDNIBBL6
1773: B7 0E     1481          ADR  RDDAEPIL
1775: C7 0E     1482          ADR  RDDAEPI1
1777: 00 00     1483          HEX  0000
                1484 ;
1779: 22 0C     1485 CSLM0    ADR  WRS1
177B: 25 0C     1486          ADR  WRS2
177D: 32 0C     1487          ADR  WRS3
177F: 35 0C     1488          ADR  WRS4
1781: 4F 0C     1489          ADR  WRS5
1783: 52 0C     1490          ADR  WRS6
1785: 67 0C     1491          ADR  WRS7
1787: 6A 0C     1492          ADR  WRS8
1789: 85 0C     1493          ADR  WRS9
178B: 88 0C     1494          ADR  WRS10
178D: 91 0C     1495          ADR  WRS11
178F: 94 0C     1496          ADR  WRS12
1791: C8 0C     1497          ADR  WRS13
1793: CB 0C     1498          ADR  WRS14
1795: F0 0C     1499          ADR  GAP2
1797: F3 0C     1500          ADR  WRS15
1799: 1D 0D     1501          ADR  WRS16
179B: 20 0D     1502          ADR  WRS17
179D: 35 0D     1503          ADR  WRS18
179F: 38 0D     1504          ADR  WRS19
17A1: 4D 0D     1505          ADR  WRS20
17A3: 50 0D     1506          ADR  WRS21
17A5: 64 0D     1507          ADR  WRS22
17A7: 67 0D     1508          ADR  WRS23
17A9: 82 0D     1509          ADR  WRS24
17AB: 85 0D     1510          ADR  WRS25
17AD: B6 0D     1511          ADR  WRS26
17AF: B9 0D     1512          ADR  WRS27
17B1: C2 0D     1513          ADR  WRS28
17B3: C5 0D     1514          ADR  WRS29
17B5: DD 0D     1515          ADR  WRS30
17B7: E0 0D     1516          ADR  WRS31
17B9: 00 00     1517          HEX  0000
                1518 ;
                1519 ;
                1520 ; Tabellen anlegen
                1521 ; Schreibübersetzungstab. 4-mal kopieren
                1522 ;
17BB: A2 00     1523 MAKETABS LDX  #$00
17BD: 8A        1524 TABLOOP  TXA             ;800-83F:00
17BE: 29 C0     1525          AND  #$C0       ;840-87F:10
17C0: 4A        1526          LSR             ;880-8BF:20
17C1: 4A        1527          LSR             ;8C0-8FF:30
```

```
17C2: 9D 00 08  1528          STA  WRTRANS1,X ;900-93F:00
17C5: 4A        1529          LSR             ;940-97F:04
17C6: 4A        1530          LSR             ;980-9BF:08
17C7: 9D 00 09  1531          STA  WRTRANS2,X ;9C0-9FF:0C
17CA: 4A        1532          LSR             ;A00-A3F:00
17CB: 4A        1533          LSR             ;A40-A7F:01
17CC: 9D 00 0A  1534          STA  WRTRANS3,X ;A80-ABF:02
17CF: E8        1535          INX             ;AC0-AFF:03
17D0: D0 EB     1536          BNE  TABLOOP
                1537 ;
17D2: A2 3F     1538          LDX  #$3F
17D4: BD 56 0F  1539 LOOP2    LDA  SUET,X
17D7: 9D 00 0B  1540          STA  WRTRANS4,X ;B00-B3F
17DA: 9D 40 0B  1541          STA  WRTRANS4+$40,X ;B40-B7F
17DD: 9D 80 0B  1542          STA  WRTRANS4+$80,X ;B80-BBF
17E0: 9D C0 0B  1543          STA  WRTRANS4+$C0,X ;BC0-BFF
17E3: 8A        1544          TXA
17E4: 0A        1545          ASL
17E5: 0A        1546          ASL
17E6: 48        1547          PHA             ;300-30F:00
17E7: 29 C0     1548          AND  #$C0       ;310-31F:40
17E9: 9D 00 03  1549          STA  RDTRANS1,X ;320-32F:80
17EC: 68        1550          PLA             ;330-33F:C0
17ED: 0A        1551          ASL             ;340-343:00
17EE: 0A        1552          ASL             ;344-347:40
17EF: 48        1553          PHA             ;348-34B:80
17F0: 29 C0     1554          AND  #$C0       ;34C-34F:C0
17F2: 9D 40 03  1555          STA  RDTRANS2,X ;   ...
17F5: 68        1556          PLA             ;37C-37F:C0
17F6: 0A        1557          ASL             ;380:00
17F7: 0A        1558          ASL             ;381:40
17F8: 9D 80 03  1559          STA  RDTRANS3,X ;382:80
17FB: CA        1560          DEX             ;383:C0
17FC: 10 D6     1561          BPL  LOOP2      ;   ...
17FE: 60        1562          RTS             ;3BF:C0
                1563 ;
                1564 ;
                1565 ; Hi-Byte vom Speicher für jeden Track
                1566 ;
17FF: E0 D0 B0  1567 PUFTAB   HEX  E0D0B0A090807060504030 20 ;
1802: A0 90 80 70 60 50 40 30
180A: 20
                1568 ;
```

6502 / 65C02-Tabelle

Function	Mnemon.	Akku		Immedi.		Zero-P.		Zero, X		Zero, Y		Absol.		Abs, X		Abs, Y		(Ind, X)		(Ind), Y		Implizit		Relativ		(Ind)		Status
		Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	N V D I Z C
Load	LDA			A9	2	A5	3	B5	4			AD	4	BD	4*	B9	4*	A1	6	B1	5*							N - - - Z -
	LDX			A2	2	A6	3			B6	4	AE	4			BE	4*											N - - - Z -
	LDY			A0	2	A4	3	B4	4			AC	4	BC	4*													N - - - Z -
Store	STA					85	3	95	4			8D	4	9D	5	99	5	81	6	91	6					92	5	- - - - - -
	STX					86	3			96	4	8E	4															- - - - - -
	STY					84	3	94	4			8C	4															- - - - - -
Transfer	TAX																					AA	2					N - - - Z -
	TXA																					8A	2					N - - - Z -
	TAY																					A8	2					N - - - Z -
	TYA																					98	2					N - - - Z -
Stack Ptr	TSX																					BA	2					N - - - Z -
	TXS																					9A	2					- - - - - -
Stack	PHA																					48	3					- - - - - -
	PLA																					68	4					N - - - Z -
Status R.	PHP																					08	3					- - - - - -
	PLP																					28	4					N V D I Z C
Flags	CLC																					18	2					- - - - - 0
	SEC																					38	2					- - - - - 1
	CLI																					58	2					- - - 0 - -
	SEI																					78	2					- - - 1 - -
	CLD																					D8	2					- - 0 - - -
	SED																					F8	2					- - 1 - - -
	CLV																					B8	2					- 0 - - - -
Jump	JMP											4C	3					*7C*	*6'*							*6C*	*5/6'*	- - - - - -
	JSR											20	6															- - - - - -
Return	RTS																					60	6					- - - - - -
	RTI																					40	6					N V D I Z C
Compare	CMP			C9	2	C5	3	D5	4			CD	4	DD	4*	D9	4*	C1	6	D1	5*					*D2*	*5*	N - - - Z C
	CPX			E0	2	E4	3					EC	4															N - - - Z C
	CPY			C0	2	C4	3					CC	4															N - - - Z C
	BIT			*89*	*2*	24	3	*34*	*4*			2C	4	*3C*	*4*													7 6 - - Z -
Branch	BMI																							30	2+			- - - - - -
N = 0	BPL																							10	2+			- - - - - -
	BEQ																							F0	2+			- - - - - -
Z = 0	BNE																							D0	2+			- - - - - -
	BCS																							B0	2+			- - - - - -
C = 0	BCC																							90	2+			- - - - - -
	BVS																							70	2+			- - - - - -
V = 0	BVC																							50	2+			- - - - - -
Increment	INC	*1A*	*2*			E6	5	F6	6			EE	6	FE	*7/6*													N - - - Z -
	INX																					E8	2					N - - - Z -
	INY																					C8	2					N - - - Z -
Decrement	DEC	*3A*	*2*			C6	5	D6	6			CE	6	DE	*7/6*													N - - - Z -
	DEX																					CA	2					N - - - Z -
	DEY																					88	2					N - - - Z -
Add /Subt.	ADC			69	2	65	3	75	4			6D	4	7D	4*	79	4*	61	6	71	5*					*72*	*5*	N V - - Z C
	SBC			E9	2	E5	3	F5	4			ED	4	FD	4*	F9	4*	E1	6	F1	5*					*F2*	*5*	N V - - Z C
Boolean	AND			29	2	25	3	35	4			2D	4	3D	4*	39	4*	21	6	31	5*					*32*	*5*	N - - - Z -
	ORA			09	2	05	3	15	4			0D	4	1D	4*	19	4*	01	6	11	5*					*12*	*5*	N - - - Z -
	EOR			49	2	45	3	55	4			4D	4	5D	4*	59	4*	41	6	51	5*					*52*	*5*	N - - - Z -
Shift	ASL	0A	2			06	5	16	6			0E	6	1E	*7/6*													N - - - Z C
	LSR	4A	2			46	5	56	6			4E	6	5E	*7/6*													0 - - - Z C
Rotate	ROL	2A	2			26	5	36	6			2E	6	3E	*7/6*													N - - - Z C
	ROR	6A	2			66	5	76	6			6E	6	7E	*7/6*													N - - - Z C
Sonstiges	NOP																					EA	2					- - - - - -
	BRK																					00	7					- - - 1 - -
65C02	*BRA*																							*80*	*3+*			- - - - - -
	PHX																					*DA*	*3*					- - - - - -
	PHY																					*5A*	*3*					- - - - - -
	PLX																					*FA*	*4*					N - - - Z -
	PLY																					*7A*	*4*					N - - - Z -
	STZ					*64*	*3*	*74*	*4*			*9C*	*4*	*9E*	*5*													- - - - - -
	TRB					*14*	*5*					*1C*	*6*															7 6 - - Z -
	TSB					*04*	*5*					*0C*	*6*															7 6 - - Z -
Byteanzahl		1		2		2		2		2		3		3		3		2		2		1		2		2		

* = 1 Takt mehr bei Seitenübergang.
\+ = 2 Takte bei Nicht-Verzweigung, 3 Takte bei Verzweigung, 4 Takte bei Verzweigung mit Seitenübergang.
' = JMP ($HHLL) und JMP ($HHLL, X) je 3 Bytes; passen nicht in Systematik
kursiv = 65C02-Änderungen

Anhang 2: ASCII-Tabelle

NUL	@ §	00	00000000	000	@ §	40	01000000	064	@ §	80	10000000	128	@ §	C0	11000000	192
	A	01	00000001	001	A	41	01000001	065	A	81	10000001	129	A	C1	11000001	193
	B	02	00000010	002	B	42	01000010	066	B	82	10000010	130	B	C2	11000010	194
	C	03	00000011	003	C	43	01000011	067	C	83	10000011	131	C	C3	11000011	195
EOT	D	04	00000100	004	D	44	01000100	068	D	84	10000100	132	D	C4	11000100	196
	E	05	00000101	005	E	45	01000101	069	E	85	10000101	133	E	C5	11000101	197
	F	06	00000110	006	F	46	01000110	070	F	86	10000110	134	F	C6	11000110	198
BELL	G	07	00000111	007	G	47	01000111	071	G	87	10000111	135	G	C7	11000111	199
←	H	08	00001000	008	H	48	01001000	072	H	88	10001000	136	H	C8	11001000	200
TAB	I	09	00001001	009	I	49	01001001	073	I	89	10001001	137	I	C9	11001001	201
↓	J	0A	00001010	010	J	4A	01001010	074	J	8A	10001010	138	J	CA	11001010	202
↑	K	0B	00001011	011	K	4B	01001011	075	K	8B	10001011	139	K	CB	11001011	203
FF	L	0C	00001100	012	L	4C	01001100	076	L	8C	10001100	140	L	CC	11001100	204
RTN	M	0D	00001101	013	M	4D	01001101	077	M	8D	10001101	141	M	CD	11001101	205
SO	N	0E	00001110	014	N	4E	01001110	078	N	8E	10001110	142	N	CE	11001110	206
SI	O	0F	00001111	015	O	4F	01001111	079	O	8F	10001111	143	O	CF	11001111	207
	P	10	00010000	016	P	50	01010000	080	P	90	10010000	144	P	D0	11010000	208
XON	Q	11	00010001	017	Q	51	01010001	081	Q	91	10010001	145	Q	D1	11010001	209
	R	12	00010010	018	R	52	01010010	082	R	92	10010010	146	R	D2	11010010	210
XOFF	S	13	00010011	019	S	53	01010011	083	S	93	10010011	147	S	D3	11010011	211
	T	14	00010100	020	T	54	01010100	084	T	94	10010100	148	T	D4	11010100	212
→	U	15	00010101	021	U	55	01010101	085	U	95	10010101	149	U	D5	11010101	213
	V	16	00010110	022	V	56	01010110	086	V	96	10010110	150	V	D6	11010110	214
	W	17	00010111	023	W	57	01010111	087	W	97	10010111	151	W	D7	11010111	215
	X	18	00011000	024	X	58	01011000	088	X	98	10011000	152	X	D8	11011000	216
	Y	19	00011001	025	Y	59	01011001	089	Y	99	10011001	153	Y	D9	11011001	217
	Z	1A	00011010	026	Z	5A	01011010	090	Z	9A	10011010	154	Z	DA	11011010	218
ESC	[Ä	1B	00011011	027	[Ä	5B	01011011	091	[Ä	9B	10011011	155	[Ä	DB	11011011	219
	\ Ö	1C	00011100	028	\ Ö	5C	01011100	092	\ Ö	9C	10011100	156	\ Ö	DC	11011100	220
	] Ü	1D	00011101	029	] Ü	5D	01011101	093	] Ü	9D	10011101	157	] Ü	DD	11011101	221
	↑	1E	00011110	030	↑	5E	01011110	094	↑	9E	10011110	158	↑	DE	11011110	222
	_	1F	00011111	031	_	5F	01011111	095	_	9F	10011111	159	_	DF	11011111	223
		20	00100000	032	'	60	01100000	096		A0	10100000	160	'	E0	11100000	224
	!	21	00100001	033	a	61	01100001	097	!	A1	10100001	161	a	E1	11100001	225
	"	22	00100010	034	b	62	01100010	098	"	A2	10100010	162	b	E2	11100010	226
	#	23	00100011	035	c	63	01100011	099	#	A3	10100011	163	c	E3	11100011	227
	$	24	00100100	036	d	64	01100100	100	$	A4	10100100	164	d	E4	11100100	228
	%	25	00100101	037	e	65	01100101	101	%	A5	10100101	165	e	E5	11100101	229
	&	26	00100110	038	f	66	01100110	102	&	A6	10100110	166	f	E6	11100110	230
	'	27	00100111	039	g	67	01100111	103	'	A7	10100111	167	g	E7	11100111	231
	(	28	00101000	040	h	68	01101000	104	(	A8	10101000	168	h	E8	11101000	232
	)	29	00101001	041	i	69	01101001	105	)	A9	10101001	169	i	E9	11101001	233
	*	2A	00101010	042	j	6A	01101010	106	*	AA	10101010	170	j	EA	11101010	234
	+	2B	00101011	043	k	6B	01101011	107	+	AB	10101011	171	k	EB	11101011	235
	,	2C	00101100	044	l	6C	01101100	108	,	AC	10101100	172	l	EC	11101100	236
	–	2D	00101101	045	m	6D	01101101	109	–	AD	10101101	173	m	ED	11101101	237
	.	2E	00101110	046	n	6E	01101110	110	.	AE	10101110	174	n	EE	11101110	238
	/	2F	00101111	047	o	6F	01101111	111	/	AF	10101111	175	o	EF	11101111	239
	0	30	00110000	048	p	70	01110000	112	0	B0	10110000	176	p	F0	11110000	240
	1	31	00110001	049	q	71	01110001	113	1	B1	10110001	177	q	F1	11110001	241
	2	32	00110010	050	r	72	01110010	114	2	B2	10110010	178	r	F2	11110010	242
	3	33	00110011	051	s	73	01110011	115	3	B3	10110011	179	s	F3	11110011	243
	4	34	00110100	052	t	74	01110100	116	4	B4	10110100	180	t	F4	11110100	244
	5	35	00110101	053	u	75	01110101	117	5	B5	10110101	181	u	F5	11110101	245
	6	36	00110110	054	v	76	01110110	118	6	B6	10110110	182	v	F6	11110110	246
	7	37	00110111	055	w	77	01110111	119	7	B7	10110111	183	w	F7	11110111	247
	8	38	00111000	056	x	78	01111000	120	8	B8	10111000	184	x	F8	11111000	248
	9	39	00111001	057	y	79	01111001	121	9	B9	10111001	185	y	F9	11111001	249
	:	3A	00111010	058	z	7A	01111010	122	:	BA	10111010	186	z	FA	11111010	250
	;	3B	00111011	059	ä	7B	01111011	123	;	BB	10111011	187	ä	FB	11111011	251
	<	3C	00111100	060	ö	7C	01111100	124	<	BC	10111100	188	ö	FC	11111100	252
	=	3D	00111101	061	ü	7D	01111101	125	=	BD	10111101	189	ü	FD	11111101	253
	>	3E	00111110	062	~ ß	7E	01111110	126	>	BE	10111110	190	~ ß	FE	11111110	254
	?	3F	00111111	063	DEL	7F	01111111	127	?	BF	10111111	191	DEL	FF	11111111	255

Anhang 3: Bildschirmdarstellung 1. Zeichensatz

Zeichen	Darstellung		
	Nor	Fls	Inv
	A0	60	20
!	A1	61	21
“	A2	62	22
#	A3	63	23
$	A4	64	24
%	A5	65	25
&	A6	66	26
’	A7	67	27
(	A8	68	28
)	A9	69	29
*	AA	6A	2A
+	AB	6B	2B
,	AC	6C	2C
–	AD	6D	2D
.	AE	6E	2E
/	AF	6F	2F
0	B0	70	30
1	B1	71	31
2	B2	72	32
3	B3	73	33
4	B4	74	34
5	B5	75	35
6	B6	76	36
7	B7	77	37
8	B8	78	38
9	B9	79	39
:	BA	7A	3A
;	BB	7B	3B
<	BC	7C	3C
=	BD	7D	3D
>	BE	7E	3E
?	BF	7F	3F

Zeichen	Darstellung		
	Nor	Fls	Inv
@ §	C0	40	00
A	C1	41	01
B	C2	42	02
C	C3	43	03
D	C4	44	04
E	C5	45	05
F	C6	46	06
G	C7	47	07
H	C8	48	08
I	C9	49	09
J	CA	4A	0A
K	CB	4B	0B
L	CC	4C	0C
M	CD	4D	0D
N	CE	4E	0E
O	CF	4F	0F
P	D0	50	10
Q	D1	51	11
R	D2	52	12
S	D3	53	13
T	D4	54	14
U	D5	55	15
V	D6	56	16
W	D7	57	17
X	D8	58	18
Y	D9	59	19
Z	DA	5A	1A
[Ä	DB	5B	1B
/ Ö	DC	5C	1C
] Ü	DD	5D	1D
^	DE	5E	1E
_	DF	5F	1F

Zeichen	Darstellung Crtl
@	80
A	81
B	82
C	83
D	84
E	85
F	86
G	87
H	88
I	89
J	8A
K	8B
L	8C
M	8D
N	8E
O	8F
P	90
Q	91
R	92
S	93
T	94
U	95
V	96
W	97
X	98
Y	99
Z	9A
[	9B
\	9C
]	9D
^	9E
–	9F

Zeichen IIplus	Zeichen IIe/IIc	Darstellung Nor
		E0
!	a	E1
“	b	E2
#	c	E3
$	d	E4
%	e	E5
&	f	E6
’	g	E7
(	h	E8
)	i	E9
*	j	EA
+	k	EB
,	l	EC
–	m	ED
.	n	EE
/	o	EF
0	p	F0
1	q	F1
2	r	F2
3	s	F3
4	t	F4
5	u	F5
6	v	F6
7	w	F7
8	x	F8
9	y	F9
:	z	FA
;	{ ä	FB
<	| ö	FC
=	} ü	FD
>	~ ß	FE
?	rub	FF

Der zweite Zeichensatz des IIc und des enhanced IIe enthält von $40–$5F Mauszeichen und von $60–$7F den inversen Kleinbuchstaben-Zeichensatz.

Stichwortverzeichnis

alphabetisches Verzeichnis

numerisches Verzeichnis

Programm-Verzeichnis

Hüthig

Arne Schäpers

ProDOS-Analyse

Versionen 1.0.1, 1.0.2, 1.1.1

1985, 470 S., kart., DM 68,—
ISBN 3-7785-1134-3

„Die ProDOS Analyse" ist die umfangreichste und detaillierteste Darstellung, die jemals ein Apple-Betriebssystem erfahren hat.

Wer die „Innereien" von ProDOS bis zum letzten Byte, z. T. bis ins letzte Bit kennenlernen möchte, braucht dieses Buch. Das komplette Betriebssystem (Urlader, MLI, Disk-Driver, RAM-Disk-Driver und Uhr-Routine) mit Ausnahme des BASIC-SYSTEM wird mit umfangreichen Kommentaren und Übersichtstabellen disassembliert. Dabei werden alle bisherigen Versionen von 1.0.1 bis 1.1.1 berücksichtigt.

„Die ProDOS Analyse" beschreibt erstmals auch mehrere Programmfehler, die bis in die neueste Version zu finden sind. Auch die nicht im „Technical Reference Manual" aufgeführten Eigenschaften von ProDOS werden analysiert und beschrieben, z. B. die vertrackten eingebauten Testroutinen zur Identifikation der verschiedenen Apple II Modelle und eventueller Nachbaugeräte. Programmierer, die ProDOS versionsabhängig „patchen" möchten, erhalten hier den genauen Überblick, wo was geändert werden muß, damit dies keine negativen Konsequenzen hat. Durch die minutiöse theoretische Sezierung von ProDOS eröffnen sich völlig neue programmierpraktische Perspektiven.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Ulrich Stiehl

Hüthig

Apple-Assembler

1984, 200 S., 3 Abb., kart.,
DM 34,—
ISBN 3-7785-1047-9

Begleitdiskette DM 28,—
ISBN 3-7785-1048-7

„Apple Assembler" wendet sich an alle, die bereits Anfängerkenntnisse der 6502-Programmierung haben - z. B. aufgrund des Buches „Apple Maschinensprache" - und nunmehr ein Nachschlagewerk für ihren Apple II Plus / IIe / IIc suchen, in dem alle wichtigen ROM-Routinen sowie eine Vielzahl sonstiger Hilfsprogramme in seiner systematischen Form zusammengestellt werden. Insgesamt umfaßt dieses Buch über 40 Utilities, darunter mehrere völlig neuartige Programme wie Double-Lores, Double Hilres, Screen-Format u.a.

Der erste Teil enthält ein Repetorium der wichtigsten Befehle, Adressierungsarten und sonstigen Besonderheiten des 6502.

Im zweiten Teil werden alle Adressen des Monitors zusammengestellt, die für Assembler-Programmierer von Nutzen sein können. Darüber hinaus findet der Leser Unterroutinen für hexadezimale Addition / Subtraktion / Multiplikation / Division, Binär-Hex-ASCII-Umwandlung usw.

Der dritte Teil befaßt sich mit der Speicherverwaltung der Language Card und der IIe-64K-Karte und enthält Move-Programme zum Verschieben von Daten in die und aus der Language Card sowie der 64K-Karte.

Der vierte Teil ist dem Applesoft-ROM gewidmet und listet eine große Anzahl nützlicher Interpreter-Adressen. Bei den Utility-Programmen liegt das Schwergewicht auf Fließkommamathematik einschließlich Print Using.

Der letzte Teil behandelt den Text- und Graphikspeicher. Neben einem professionellen Maskengeneratorprogramm werden auch Routinen zur Double-Lores und Double-Hires-Grafik vorgestellt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Jürgen Kehrel

Hüthig

Apple-Assembler lernen

Band 1: Einführung in die Assembler-Programmierung des 6502 /65C02

1985, 234 S., kart., DM 38,—
ISBN 3-7785-1151-3

Begleitdiskette: DM 44,—
ISBN 3-7785-1243-9

Das zweibändige Werk „Apple Assembler lernen" ist ein kompletter Kurs in der Assemblerprogrammierung des 6502 und des 65C02 auf dem Apple II, der nicht da aufhört, wo andere Einführungen auf „weiterführende Literatur" verweisen. Starkes Gewicht wird auf die praktische Anwendung gelegt. Deshalb gehört zum Kurs ein vollwertiger 2-Pass Assembler, der als einer von wenigen die erweiterten Befehle der neuen IIc und IIe Prozessoren 65C02 verarbeitet und der auch lange Programme von mehr als 1 000 Zeilen in wenigen Sekunden übersetzt. Zu seinen professionellen Eigenschaften gehören neben 15 Pseudo-Opcodes, die die Arbeit mit Strings und Tabellen zu einem Kinderspiel werden lassen, ein Zeileneditor mit viel Komfort und die Fähigkeit, Quellcode in verschiedenen Formaten zu schreiben und zu lesen. Ein interaktiver Debugger und Simulator hilft Ihnen, eigene und fremde Maschinenprogramme zu verstehen. Mit seinen vielfältigen und mächtigen Möglichkeiten läßt er Sie hinter die Kulissen Ihres Rechners schauen. Sie können „sehen", was abläuft, Ihre Vorstellungskraft wird angeregt und nicht nur einfach Ihr Gedächtnis strapaziert.

Alle Programme sind 100 % kompakter Maschinencode, nicht einfach compiliertes Basic.

Im ersten Band erlernen Sie innerhalb von 35 Lektionen sämtliche Maschinenbefehle des Apple und die wichtigen Grundalgorithmen. Der Umgang mit dem Assembler und dem Simulator wird geübt, und zum Nachschlagen steht eine ausführliche Befehlsbeschreibung mit vielen praktischen Beispielen bereit.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Heinrich Kersten

Hüthig

Apple-CP/M: Assembler-Programmierung

für Einsteiger, Fortgeschrittene und 6502-Kenner

1986, ca. 200 S., kart.,
ca. DM 38,—
ISBN 3-7785-1379-6

Begleitdiskette: DM 44,—
ISBN 3-7785-1380-X

Das Buch beschreibt die Assembler-Programmierung im Rahmen des CP/M-Betriebssystems. Es wurde vorrangig für die Besitzer von Apple II-Geräten konzipiert. Der überwiegende Teil des Buches ist aber geräteunabhängig und somit auch für die Benutzer anderer CP/M-tüchtiger Fabrikate (mit 8080/Z80-Prozessor) von Interesse.

Einsteiger erhalten im ersten Abschnitt einen Intensiv-Kurs (auch in der DDT-Anwendung). Für Kenner der 6502-Programmierung werden viele Querverweise vorgestellt. Der Z80-Befehlsvorrat wird im Detail erläutert. Es folgen wichtige Basisprogramme (Konvertierungen, Binär- und BCD-Arithmetik), umfassende Diskussionen der Debugger, Assembler und Linker sowie des CP/M-Betriebssystems (Speicherverwaltung, BDOS und BIOS-Aufrufe, Bildschirmfunktionen). Den Abschluß bilden lauffähige Beispielprogramme und Utilities und ein umfangreicher Tabellen-Anhang.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Hans Gabriel

Das Buch zum Apple-Writer II/IIe

1986, 155 S., kart., DM 35,—
ISBN 3-7785-1234-X

Begleitdiskette DM 44,—
ISBN 3-7785-1337-0

„Das Buch zum Apple Writer II/IIe" wendet sich an alle, die dieses Textverarbeitungssystem schon einsetzen oder noch einsetzen wollen.

In einzelnen, in sich geschlossenen Kapiteln erlernt der Leser alle Funktionen und erzielt schnelle Erfolgserlebnisse. Im ersten Kapitel werden typische Arbeitsstellungen der Textverarbeitung und ihre systematische Bearbeitung vorgestellt. Das zweite Kapitel stellt in logischen Funktionsgruppen die Befehle vor, mit denen der Applewriter während der Textarbeit direkt gesteuert werden kann. Kapitel 3 zeigt die Programmierung des Applewriters in der Text-Kommando-Sprache TKS (WPL). Dazu werden Beispiele analysiert. Auf der Begleitdiskette zum Buch, die seperat bezogen werden kann, sind darüber hinaus umfangreiche Schwerpunkterklärungen enthalten, die über die Help-Funktion vom Benutzer abgerufen werden können. Eine kleine Adreßdatenbank mit der dazugehörigen Dienstprogrammen zur Pflege der Daten und zu ihrem Einsatz in Einzel- und Serienbriefen befindet sich ebenfalls auf Diskette.

„Das Buch zum Apple Writer II/IIe" behandelt sowohl die alte Programmversion für den Apple IIplus als auch die neue Ausgabe, die 128-kByte auf dem Apple IIe oder Apple IIc unterstützt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Ulrich Stiehl

Hüthig

ProDOS für Aufsteiger

Band 1

2. geänderte Auflage 1985,
208 S., kart., DM 28,—
ISBN 3-7785-1098-3

„Apple ProDOS für Aufsteiger" ist der Nachfolgeband zu „Apple DOS 3.3 — Tips und Tricks". Applesoft-Programmierer, die unter DOS 3.3 gearbeitet haben, werden sich schnell an ProDOS gewöhnen, da ProDOS und DOS 3.3 in dieser Hinsicht weitgehend kompatibel sind. Dagegen müssen Assembler-Programmierer völlig umdenken. Deshalb liegt das Schwergewicht dieses Nachfolgebandes auf der Assemblerprogrammierung und der minutiösen Darstellung der ProDOS-internen Systemadressen, die jedoch auch für Applesoft-Programmierer von großer Bedeutung sind.
Im ersten Teil wird zunächst ein allgemeiner Überblick über das neue „Professional Disk Operation System" gegeben. Im Anschluß daran folgt eine Gegenüberstellung der Geschwindigkeit des Diskettenzugriffs. Dann wird die interne Speicherorganisation detailliert beschrieben (Boot-Vorgang, Zero-Page, ProDOS-Vektoren, Basic-System-Puffer, Basic-System-Global-Page, Basic-Command-Handler, I/O-Vektoren, ProDOS-Global-Page, Language-Card-Organisation, Interrupt, Disk-Drivewr, Reboot-Programm usw.). Ebenso ausführlich wird die externe Speicherorganisation geschildert (Spuren, Sektoren, Blocks, Directory-Struktur, Volume Bit Map, Datastrukturen usw.). Schließlich wird das MLI (Machine Language Interface) mit zahlreichen praktischen Anwendungsbeispielen erläutert. Insgesamt enthält ProDOS-Buch ca. 70 Seiten mit eigens für dieses Werk entwickelten Programmen.
Im zweiten Teil werden die Basic-System-Befehle für Applesoft-Programmierer systematisch erläutert. Allerdings wird die Kenntnis von „Apple DOS 3.3" vorausgesetzt. „ProDOS für Aufsteiger" ist deshalb nicht nur für Assembler - sondern auch für fortgeschrittene Appelsoft-Programmierer ein unentbehrliches Nachschlage-und Handbuch für die Programmierpraxis.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Das zweibändige Werk „Apple-Assembler lernen" ist ein kompletter Kursus über die Assemblerprogrammierung des 6502 und 65C02 auf dem Apple II, der nicht da aufhört, wo andere Einführungen auf „weiterführende Literatur" verweisen.

Der zweite Band stellt Programme und Unterroutinen vor, um fast alle grundlegenden Probleme auf dem Apple in Maschinensprache zu lösen: Dazu gehören Ein- und Ausgabeoperationen wie z.B. formatierte Bildschirmausgabe in 40 und 80 Z/Z., Lores- und Hires-Grafik, Hires-Schrift durch Bit-Grafik, Hires-Fenster incl. Fensterscroll. Ansprache von Diskettenlaufwerken entweder unter DOS und ProDOS oder auch ganz direkt Nibble für Nibble am Beispiel eines schnellen Kopierprogramms, das in einem Durchgang formatiert und Daten schreibt.

Zugriffe auf die Stringverwaltung und die Fließkommaarithmetik von Applesoft werden ebenso behandelt wie elegante Musik- und Toneffekte. Dabei wird ausgiebig von ROM-Routinen Gebrauch gemacht, die jeweils im Zusammenhang erklärt werden. Sie lernen viel über das Innenleben Ihres Apple und wie Sie BASIC-Programme mit Maschinensprache verbessern können.

Der zweite Band setzt die Grundkenntnis der Prozessorbefehle und den Umgang mit einem Assembler voraus. Assessor (zusammen mit IDUS auf der Begleitdiskette zu Band 1 erschienen) ist auch hier wieder die ideale Hilfe, um Assemblerprogrammierung auf dem Apple zu lernen.